Autopoietic Automata

Wiedermann, Jiří
2005

Institute of Computer Science
Academy of Sciences of the Czech Republic

Autopoietic Automata

Jiří Wiedermann

Technical report No. 929

February 2005

Pod Vodárenskou věží 2, 182 07 Prague 8, phone: (+420) 266 053 520, fax: (+4202) 86 585 789,
e-mail:jiri.wiedermann@cs.cas.cz

Institute of Computer Science
Academy of Sciences of the Czech Republic

Autopoietic Automata[1]

Jiří Wiedermann

Technical report No. 929

February 2005

Abstract:

We introduce a new formal computational model designed for studying the information transfer among the generations of self–reproducing machines — so–called autopoietic automata. These can be seen as finite state transducers whose "program" can become a subject of their own processing. An autopoietic automaton can algorithmically generate an offspring controlled by a program which is a modification of its parent's program. We show that the computational power of lineages of autopoietic automata is equal to that of an interactive nondeterministic Turing machine. We also prove that there exists an autopoietic automaton giving rise to an unlimited evolution, providing suitable inputs are delivered to individual automata. However, the problem of a sustainable evolution, asking for an arbitrary autopoietic automaton and arbitrary inputs whether there is an infinite lineage of its offspring is undecidable.

Keywords:
self–reproducing automaton; interactive Turing machine; evolution

[1]This research was partially supported by grant No. 1ET100300517 within the National Research Program "Information Society".

1 Introduction

The notion of autopoiesis has been coined by Chilean biologists Maturana and Varela since the nineteen seventies. Literally, autopoiesis means self–production and denotes a process whereby a system (or an "organization", as Maturana and Varela call it) produces itself (for more details concerning computational autopoiesis, cf. [2]). Autopoiesis, as its proponents understand it, is not a precisely, mathematically or otherwise formally defined notion and in fact we will use this notion in its literal meaning to denote self–producing or self–creating units. The reason for calling our model autopoietic automata has been the aspiration to distinguish such automata by name from the notoriously known self–reproducing automata which are a kind of cellular automata. The autopoietic automata are not based on the formalism of cellular automata and in fact they build upon the classical models of the finite state automata. They are designed as a mathematical model of self-evolving units capturing the computational (or information processing) essence of self–production, i.e., the use of a "program" both to drive the (computational) behavior of a unit and to serve as a "template" for the evolutionary process. The lastly mentioned idea originates, of course, from von Neumann [5] whose stress in designing his self–reproducing automata had been just on the design of mechanisms of self–reproduction alone. In our modelling, however, we will not be concerned in these mechanisms: rather, we take their existence as granted and we concentrate instead on algorithms controlling the variations in the reproduction process and (hence) the ("genetic") information transfer from the parental machine to its offspring. That is, we will not be interested in producing exact copies of the parental machine: in our modelling we will focus on the evolution in which offsprings possess qualities different fom their parents. Thus, instead of self–reproduction we should rather speak more precisely about self–like production.

In von Neumann's seminal paper on self–reproducing automata, the problem of the variation of genetic information was not the main issue. Nevertheless, a related question concerning the "evolutionary growth of complexity" of self–reproducing automata has become the issue in the field of artificial life (for an overview, see [1]). In the absence of suitable computational models neither this question nor the related problem of the computational power of automata exhibiting the evolutionary growth of their complexity could have been answered convincingly.

In this paper we present a computational model answering the previous questions. Our model is inspired by contemporary cellular biology. In its design it abstracts the information processing, reproducing and evolutional abilities of the living cells. An autopoietic automaton is a specific kind of a nondeterministic finite state transducer which has access to the representation of its own transition relation. Controlled by this transition relation and making use of the possibility to read the representation of this relation, an autopoietic automaton computes and outputs the transition relation of its offspring. In this way the changes in the new transition relation are controlled by the parental machine. Our main result shows that a series of lineal descents of a single autopoietic automaton (a lineage of autopoietic automata) has a notable computational power — the same as an interactive nondeterministic Turing machine. We also construct an autopoietic automaton which generates a lineage containing all autopoietic automata, i.e., the members of this lineage exhibit unbounded growth of complexity in the computational sense. Within our model, this result answers positively the related question asked by McMullin and its predecessors in the field of artificial life (cf. [1]). Finally, we define the problem of the so–called sustainable evolution which asks after any autopoietic automaton and any infinite sequence of inputs whether there is an infinite lineage generated by that automaton on that input. We show that this is an undecidable problem.

The content of the paper is as follows. Section 2 contains the formal definition of an autopoietic automaton and of its computations, too. In Section 3 the computational power of lineage of autopoietic automata is characterized via interactive nondeterministic Turing machines. The computational aspects of the evolution of autopoietic automata, especially the unboundedness of evolutionary complexity growth and an evolution's sustainability, are studied in Section 4. Section 5 recapitulates the main contributions of the paper.

It is very tempting to interpret the previous results in the framework of original scientific disciplines which served as inspiration for our modelling. As to the extent to which our model captures the information processing and evolutionary abilities of living cells, our results seem to be among the first

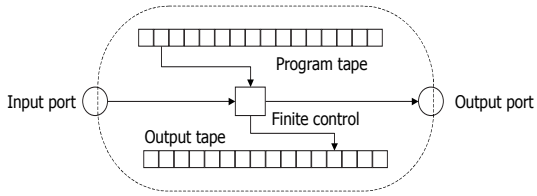

Figure 2.1: The schema of an autopoietic automaton

formal results shedding light on the computational nature and power of the respective mechanisms.

2 Autopoietic automata

Autopoietic automata are nondeterministic finite state machines capturing the information processing, reproducing and evolving abilities of living cells. Technically, an autopoietic automaton is a nondeterministic transducer (a Mealy automaton) computing and outputting the transition relation of its offspring. The design of an autopoietic automaton supports working in two modes. The first of them is a standard transducer mode controlled by a transition relation and processing external input information read through an input port. In this phase the results of a computation (if any) are sent to the output port. The second mode is a reproducing mode which is controlled by the same transition relation as before. This time, however, no external information is taken into account and, instead, the representation of the transition relation itself is used as a kind of the internal input. For this purpose the representation of the automaton's own transition relation is available to an autopoietic automaton on a special, so–called program tape. It is a two–way read–only tape. The result of the reproducing mode is written on a special one–way write–only output tape. Of course, both tapes mentioned before are finite. After finishing the reproduction, the information written on this tape is interpreted as a transition relation of a new autopoietic automaton and the tape itself becomes a new automaton's program tape. The new automaton starts its activity with the empty output tape. The new automaton is seen as an offspring of the original automaton. Depending on the transition relation of the parental automaton, the transition function of the new automaton can differ from the original transition relation. Schematically, the architecture of an autopoietic automaton is depicted in Fig. 1.

Now we are ready to proceed to a formal definition of an autopoietic automaton. One of our final aims is to study the sequences of such automata with an increasing number of internal states and working over alphabets of increasing size. Therefore, in general we will consider an infinite set Q of states whose members will be numbered, i.e., $Q = \{q_1, q_2, \ldots\}$ and similarly an infinite, so–called external working alphabet $\Sigma = \{\sigma_1, \sigma_2, \ldots\}$. An equivalent, so–called *internal representation* of the members of these sets will be via sequences of zeros: the i–th member will be encoded as a sequence of i zeros, abbreviated as 0^i.

Definition 2.1 *An* autopoietic automaton *is a six–tuple* $A = \{\Sigma, Q, R, q_1, q_2, \delta\}$, *where*

- Σ, *with* $\varepsilon \in \Sigma$, *is the finite or infinite* external alphabet *whose symbols are read on the input port or are written to the output port, one symbol at a time;* ε *is the empty symbol;*
- Q *is the finite or infinite* set of states*;*
- $R \subset Q$, $R \neq \emptyset$ *is the distinguished* set of reproducing states*;*
- $q_1 \in Q - R$ *is the* initial state *in which the computation of* A *starts, with either head at the left end on the respective tape;*
- $q_2 \in R$ *is the* final reproducing state*; entering it finishes the reproduction mode of* A *and starts a computation of the* A*'s offspring which is an autopoietic automaton whose transition relation*

had been generated by A on its output tape. Simultaneously, A empties its output tape and restarts its computation from its initial state. Since this time on the computation of both A and that of its offspring have been independent, each of them receiving its own input;

- δ, *the* transition relation, *comes in two forms depending on whether $q \in Q - R$ or $q \in R$:*
  - Transducer mode*: if A is in state $q \in Q - R$, then we say that A is in a transducer mode. Then δ is a finite subset of $\Sigma \times Q \times \Sigma \times Q \times D$, where $D = \{d_1, d_2, d_3\}$ is the* alphabet of directions*; the elements of δ are formed by five–tuples of form $(\sigma_i, q_j, \sigma_k, q_\ell, d_m) \in \Sigma \times Q \times \Sigma \times Q \times D$ from which a sequence (in arbitrary order) is formed and encoded in binary on the program tape; the respective encoding for the above mentioned tuple is $10^i10^j10^k10^\ell10^m1$; the encoding of all tuples representing δ is also embraced by 1s (i.e., the entire encoding starts and ends by two consecutive 1s). A tuple $(\sigma_i, q_j, \sigma_k, q_\ell, d_m) \in \Sigma \times Q \times \Sigma \times Q \times D$ corresponds to one computational step (transition) of A in the following way: if A is in state $q_j \in Q - R$ and $\sigma_i \in \Sigma$ is a symbol at the input port, A changes its state to q_ℓ and writes $\sigma_k \in \Sigma$ to its output port; the position of the head on the output tape remains unchanged while the head on the program tape shifts in direction d_m (d_1 denotes the left shift by one position, d_2 the right shift by one position, d_3 means no shift);*
  - Reproducing mode: *if A is in a state $q \in R$, then A is in a reproducing mode. In such a case δ is a finite subset of $\{0,1\} \times Q \times \{0,1\} \times Q \times D$. A tuple $(\sigma_i, q_j, \sigma_k, q_\ell, d_m) \in \{0,1\} \times Q \times \{0,1\} \times Q \times D$ is represented on the A's program tape similarly as any tuple of a transducer mode, with* 0 *representing the element* 0 *and* 00 *representing* 1. *Such a tuple corresponds to one computational step of A in the following way: if A is in state $q_j \in R$, and $\sigma_i \in \{0,1\}$, the symbol scanned by the read head on the program tape, A changes its state to q_ℓ and writes $\sigma_k \in \{0,1\}$ to its output tape; doing so the head on the program tape shifts in direction d_m and the head on the output tape shifts by one position to the right.*

Note please that we have admitted that both sets Σ and Q can be infinite sets. However, the transition relation δ must always be finite (or more precisely: a finite subset of $\Sigma \times Q \times \Sigma \times Q \times D$). This unusual arrangement allows an autopoietic automaton to generate offsprings working over larger (or different) sets of states or symbols than it was possible for the original automaton. In Section 4 we will see that this is what enables a kind of evolutionary growth of complexity of the underlying automata.

An autopoietic automaton A starts its computation in state q_1, with the head on either tape in the leftmost position. The automaton reads the symbol appearing on its input port and realizes the respective transition as described in the previous definition. In general, thanks to nondeterminism, the transition relation allows several choices for the next step. As is customary with nondeterministic computations we take the viewpoint that any choice that will eventually lead the automaton to enter the final reproducing state q_2 is a legal move.

In automaton's further activities, the general rule is that while being in non–reproducing states the automaton reads the symbols from the input port and writes the symbols to the output port, possibly moving its head along the program tape. No symbols are written to the output tape. When entering a reproducing state, instead of the external symbols the automaton reads the symbols scanned by its head on the program tape and writes the binary symbols to the output tape. Entering the final reproducing state q_2, A terminates its current activities with (the binary representation of a) new transition relation δ_{new} written on its output tape, and *reproduces by fission*, so to say. It splits into two automata (see Fig. 2): the first one is driven by the original transition relation δ (denoted as Program 1 in Fig. 2) while the other one by relation δ_{new} (denoted as Program 2). The new automata start with the empty output tapes.

Thanks to the fact that on the same inputs the final reproducing state can be achieved via several computational pathes, a single nondeterministic autopoietic automaton can produce several different offsprings, not just one. In our further considerations we will assume that all such offsprings are produced, indeed. Due to the nature of the fission mechanism one of the offsprings will be identical to its parent. We can imagine that after the fission all automata continue processing their own input symbols. In principle, we see that by iterating this scenario a potentially infinite tree of offsprings

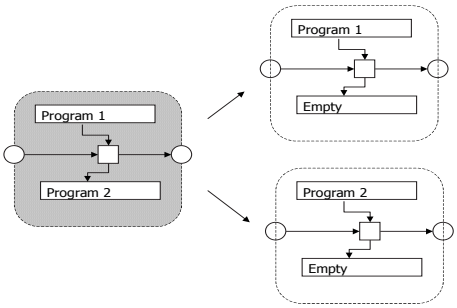

Figure 2.2: The fission of an autopoietic automaton

can be generated from a single autopoietic automaton. In this tree, each offspring is related to its parent. Now, if we concentrate to a single path starting in the root of such a tree, we get a so–called *lineage of autopoietic automata*. In its entirety each lineage realizes a translation of a potentially infinite stream of the input symbols into a similar stream of output symbols. Note that along such a computational path the automata enter the reproducing states infinitely often. In the sequel we will study the computational power of lineages.

3 The power of lineages of autopoietic automata

The first question concerning any computational model is the one of its computational power. The computational power of an autopoietic automaton is not different from that of a finite state transducer: this is because it is driven by a finite state mechanism, and its ability to read its own "program" does not add any power, since in principle the same information could be stored in the automaton's states. Nevertheless, when considering a lineage of autopoietic automata things get more interesting. We show the equivalence of lineages of autopoietic automata with so–called interactive Turing machines. This type of machines has been introduced by van Leeuwen and Wiedermann (cf. [3], [4]) when studying the so–called interactive evolutionary algorithms. Interactive Turing machines are variants of standard Turing machines adapted for processing infinite input streams. That is, instead of the input tape with a priori given input data these machines read the input data through an input port much like the autopoietic automata; the output symbols are also treated in a similar way. A nondeterministic interactive Turing machine M is said to realize a *translation* from an infinite input stream S_1 to an infinite output stream S_2 if there exists a computation of M on S_1 passing through an infinite number of accepting states of M and producing S_2 as its output (we assume that after entering an accepting state, M can prolong its computations). Two translations are considered to be *equivalent* if they are equivalent after deleting all empty symbols from them.

The equivalence between a lineage of autopoietic automata and an interactive Turing machine will be shown by mutual simulations of these devices. We start with a simpler case — namely simulating a lineage of autopoietic automata by an interactive Turing machine. Prior to proceeding to the respective simulation we must solve one fine detail. This is the problem of the unbounded input alphabet and that of translating the symbols of the external alphabet to their internal representation being used on the program tape. While the offsprings of autopoietic automata can, by their very definition, work with increasingly complex symbols, for a Turing machine with a fixed transition relation this is not possible: more complex symbols require a longer encoding. Therefore we will assume that the elements of Σ which can be directly read by the members of a lineage of autopoietic automata will be presented to a Turing machine in their unary notation, as stated in Definition 2.1. That is, for a Turing machine a symbol $\sigma_i \in \Sigma$ will be presented as a string of form 0^i and the strings in a sequence will be separated by ones. Thus, a Turing machine needs $O(i)$ steps to read (or write down) σ_i. However, we are not interested in the exact complexity of our simulations — we are merely concerned with the principal possibility of such simulations and in this respect a less efficient coding does not make any difference.

Having said so we are ready to present our first simulation theorem:

Theorem 3.1 *Any lineage of autopoietic automata can be simulated by a nondeterministic interactive Turing machine.*

Sketch of the proof. Let $\mathcal{A} = A_1, A_2, \ldots$ be a lineage of the autopoietic automata. We design a nondeterministic interactive Turing machine M simulating $\mathcal{A}$ and working as follows. M is a universal Turing machine which reads via its input port the inputs encoded in the unary notation and is able to simulate any A_i given by the description (encoding) of its transition relation δ_i. For such a purpose, M maintains the representation of both tapes of A_i on its tapes: the program tape, on which a representation of δ_i is written in the form as stated in Definition 2.1, and the output tape on which the transition relation δ_{i+1} of A_{i+1} is generated, for $i = 1, 2, \ldots$. Starting from $i = 1$, M simulates the actions of A_i as dictated by δ_i written on the program tape until A_i reproduces and the processing is taken over by A_{i+1}. In such a case, M enters its accepting state and exchanges the roles of its two tapes: the output tape becomes the program tape with δ_{i+1} already being written on it, and the original program tape after being "cleaned" becomes the new output tape. Clearly, in this way M realizes the same translation as $\mathcal{A}$ does.

□

The reverse simulation is more complicated and requires more preliminaries. First of all, we have to specify, in more detail, the Turing machine to be simulated. We will consider a nondeterministic interactive Turing machine M with one input and one output port and with only one working tape unbounded to the right. The input and output symbols will be from a finite alphabet $\Sigma_M \subset \Sigma$, with Σ being the external alphabet of the autopoietic automata at hand. The working (or tape) alphabet of M will be $\Omega = \{0, 1, \flat\}$, with $\flat$ denoting the blank symbol. The set of states of M will be $Q_M = \{q_1, q_2, \ldots, q_z\}$ for some $z > 1$. The transition relation of M will be $\delta_M \subseteq \Sigma_M \times \Omega \times Q_M \times \Sigma_M \times \Omega \times Q_M \times D$, where $D = \{d_1, d_2, d_3\}$ is the alphabet of directions of the moves of M's head on its tape and the meanings of d_i's are the same as in Definition 2.1. An element of δ_M of form $(\sigma_i, x_j, q_k, \sigma_\ell, x_m, q_n, d_p)$ is read as *"machine M, reading σ_i at its input port and scanning x_j in state q_k sends σ_ℓ to its output port, writes x_m into the scanned cell, enters state q_n and moves its head in direction d_p"*.

For the purpose of simulation, we split the processing time of M into time intervals during which the space complexity of M remains unchanged. Thanks to our assumption on the left–boundedness of M's tape, the amount of the space consumed by M's computation increases by 1 when the head of M reads $\flat$, going past the rightmost rewritten cell on its tape. Within the intervals of unchanged space complexity M is seen as a finite transducer. Let C_i be the finite transducer which is equivalent to M computing within space of size i. The idea of the simulation is then to encode the (tape) configurations of M into states of an autopoietic automaton A_i which, in its non–reproducing states, behaves as C_i. When M is to increase its space complexity, A_i switches to a reproduction mode and generates a new, "bigger" automaton A_{i+1} which in its non–reproducing states simulates C_{i+1}, etc. Thus, in fact A_i is a merger of two automata: one corresponds to C_i while the other — let us call it R — takes care of reproduction. The transition relations of both automata are written on A_i's program tape. In the reproduction mode, A_i reads the transition relation of C_i and, being controlled by R, A_i generates the code for C_{i+1} and appends to it the code of R again. Thus, the code of R remains unchanged in all A_is.

We will assume that the *tape configuration* of M is of form $\$w_1\$q\$w_2\$$, where $w_1, w_2 \in \Omega^* \cup \{\varepsilon\}$, $q \in Q_M$ and w_1 is the contents of M's tape to the left of the position of $M's$ head on M's tape, and w_2 is the contents of M's tape to the right of w_1. That is, M's head points to the first symbol of w_2 (which might be a blank symbol, $\flat$ in the case when $w_2 = \varepsilon$). The *length of the tape configuration* $\$w_1\$q\$w_2\$$ is $|w_1| + |w_2|$, i.e., the sum of lengths of w_1 and w_2, respectively.

A tape configuration of M in state q_j will be represented as a sequence $\${0, 1}^* \cup \{\varepsilon\}\$0^j\${0, 1}^* \cup \{\flat\}\$$, i.e., the states of M are represented in unary, the tape contents in binary. As mentioned above the tape configurations of M in the previous form will straightforwardly correspond to the states of an autopoietic automaton. This idea requires a slight change in the definition of an autopoietic automaton — so far the states of an autopoietic automaton have been expressed in unary on automaton's program tape. The newly proposed representation of states calls for introduction of a further separator symbol ($) among the symbols of the automaton's tape alphabet. Also note that neither the symbol read

from the input port nor the one written to the output port by M is included in the above defined notion of M's configuration — these two symbols will be represented explicitly in A_i's configuration.

Now we are ready to formulate and prove the next theorem.

Theorem 3.2 *Any nondeterministic interactive Turing machine can be simulated by a lineage of autopoietic automata.*

Sketch of the proof. Let M be a given nondeterministic interactive Turing machine. By induction on i we will construct a lineage $\{A_i\}$ of autopoietic automata such that each A_i will simulate M with tape configurations of length i (or of space complexity i) and each A_{i+1} will be an offspring of A_i, for $i \geq 2$. As mentioned above the "program" of each A_i will consist of two parts. The first part describes the transition relation of C_i while the second one that of R. For technical reasons — viz. the necessity to begin with an automaton which is able to simulate both right and left moves of the M's head on a tape of length 2 we start our induction with $i = 2$. This case is captured by C_2 and corresponds to the beginning of M's computation and to its subsequent computations until the moment when the M's head is about to enter the 3-rd cell on its tape.

Consider a generic "instruction" of δ_M of form $(\sigma_i, x_j, q_k, \sigma_\ell, x_m, q_n, d_p)$ performed over a tape configuration t_1 and resulting into tape configuration t_2, with both configurations corresponding to a tape of length 2 and the head positioned either on cell 1 or 2. This is reflected in C_2's instructions of form $(\sigma_i, t_1, \sigma_\ell, t_2, d_p)$. All these instructions, i.e., the instructions for all $\sigma_i, \sigma_\ell \in \Sigma_M$, all t_1 and t_2, all $q_k, q_n \in Q_M$ and all $d_p \in D$ conformed with δ_M are written on the A_2's program tape encoded as shown in Definition 2.1.

Any instruction of M attempting to move the M's head to the right of the 2-nd cell (or in general: increasing the current space complexity) will lead to the reproduction of A_2. Let $(\sigma_i, x_j, q_k, \sigma_\ell, x_m, q_n, d_2)$ be the move of M performed over a tape configuration t_1 of length 2 and resulting in tape configuration t_2 of length 3, with the head in t_1 positioned on the second cell and the head in t_2 positioned on the 3-rd cell.

Then, when A_2 reads σ_i in state t_1, it recognizes that this is the case when the head will move to the right. Under this circumstance A_2 will write σ_ℓ to its output port and will enter a reproducing phase in which the program tape of A_3 will be generated using the code for R. First, the program for C_3 will be constructed by reading the program for C_2. Starting from the state which corresponds to tape configuration t_2, automaton C_3 must be able to simulate all moves of M in space of size 3. In order to do so C_3 must have basically the same instructions as C_2 had. However, these instructions must be adopted for the case of the longer Turing machine tape (which is now longer by 1 cell that could contain $\flat$, 0 or 1). The "new" instructions are generated from the "old" ones by making appropriate local changes to the latter. The new instructions are generated to A_2's output tape. The respective changes must be made for all one–symbol prolongments (i.e., for 0, 1, and $\flat$) of the (current) tape configurations of M. To generate all new instructions, several (but a fixed finite number, depending on δ_M) scans over the simulation code of C_2 are needed. No doubt that this is an algorithmic procedure which can be carried by a finite automaton R thanks to the fact that the "templates" for producing new instructions are available on the program tape of A_2. An extra care is needed for capturing also transitions which so far have not been represented in C_2 since they could not come into action due to the small size of the M' tape at that time. The details of an actual design of R are left to the reader.

After the entire program for C_3 is generated, the program for R is appended to it. This finishes the generation of the entire output tape of A_2 and therefore A_2 undergoes a fission and finishes its activity. The output tape of A_2 becomes the program tape of A_3 and the simulation of M is taken over by A_3. As mentioned before, A_3 starts from the state corresponding to configuration t_2.

Now, assuming that we have any A_i for $i \geq 3$ simulating M on tape configurations of size i it is a straightforward matter to see that upon entering its reproducing phase A_i will produce A_{i+1} simulating M on tape configurations of size $i + 1$, for any $i \geq 3$. In fact, the "induction step" itself is performed by the automata themselves, by their very design as described at the beginning of this proof. The size (measured in the number of states, or length of the program tape) of our automata grows exponentially in i, i.e., in the length of the tape configuration of M at the time interval in which M is simulated by the automaton at hand.

Note that during their reproducing phases the automata in $\mathcal{A}$ are neither reading nor producing any external symbols. Therefore it is clear that on any inputs the lineage of A_is realizes the same translation as M does.

□

Putting the claims of both previous theorems together we get the following consequence:

Corollary 3.1 *The computational power of lineages of autopoietic automata is equal to the computational power of nondeterministic interactive Turing machines.*

4 The evolution of autopoietic automata

At the close of Section 2 we have already mentioned that in principle a single nondeterministic autopoietic automaton can produce an infinite tree (or at least a lineage) of its descendants. In order that this can happen, it must hold that at least one of the offsprings of a given parent must "survive", i.e., it must reach a reproducing state on a given input. Thus, in addition to the automaton's functionality the existence of infinite lineages depends on the "availability" of the "right" inputs. A trivial solution would be supplying the same inputs as before to all offsprings of an automaton which has just reproduced. Since among its offsprings there is one identical to its parent, this strategy will lead to an infinite lineage of identical automata. Obviously, this is not a very interesting case since no evolution is involved.

Under the assumption that the "right" inputs are supplied to the respective automata at each level of the descendant tree we construct an autopoietic automaton among whose descendants all possible autopoietic automata will appear. For the brevity we will call the mode of purposefully supplying inputs which will cause the given automaton to reproduce (if such inputs exist at all) the *nondeterministic input mode*. This mode assumes that the data read by the automaton exist and are supplied to the input port in such an order that will eventually lead to the fission of the automaton at hand.

Theorem 4.1 *There exists an autopoietic automaton which, when working in a nondeterministic input mode, generates a descendant tree containing all autopoietic automata.*

Sketch of the proof. The idea is to construct a single automaton which generates a descendant tree in which all autopoietic automata are enumerated. That is, this automaton will generate offsprings (direct descendants) with syntactically correct transition relations of a bounded length which will increase with the depth of the tree on which the offsprings are located. If a generated transition relation happens to be a transition relation of an autopoietic automaton that reproduces on some input, then the nondeterministic input mode will guarantee that the automaton at hand will reproduce.

Let B be the automaton we are after. Its transition relation δ will consist of k five–tuples, for some $k > 0$ (see Definition 2.1) and will be written on the B's program tape in form of a binary code. This code will read the B's program tape tuple by tuple and copy the tape either faithfully or with some modifications. Call any sequence of zeros representing a state or a symbol of Σ in the representation of the transition relation of B as written on the B's program tape, a *segment*. A segment from the program tape will be transformed via δ into a segment on the output tape according to the following rules:

- when reading a symbol in a segment, B nondeterministically decides whether to copy or skip it; in the former case, it writes 0 on the output tape and in both cases it proceeds to the next symbol in the program tape;
- after reaching the end of a segment, B will nondeterministically decide whether to prolong the segment by one additional zero;
- after processing the last segment, B will nondeterministically decide whether to add one tuple more to the generated transition relation, and if so, B will generate it nondeterministically, respecting the syntax of the encoding stated in Definition 2.1;

- in any tuple, the direction d of the move of the head on the program tape is also a subject to a nondeterministic choice.

A separator between the segments (i.e., symbol 1) will be copied without any change.

Having done so, B enters the final reproducing state. In this way, a bounded number of offsprings of B is generated, one on each branch of the respective nondeterministic computation. Each offspring possesses a syntactically correct (encoding of a) transition relation, differing in all but one offspring in certain segments from δ and containing at most one tuple more. The number of offsprings is related to the length of the program tape of B (it is exponential in k). Not all offsprings are functionally different since any transition relation admits a number of equivalent representations. On the other hand, among the offsprings there are all automata with the transition relation consisting of k or $k+1$ tuples.

Now, all these offsprings start to act on their own. Thanks to the nondeterministic input mode those automata that can in principle reach the final reproducing state will get such an input which will eventually lead to their reproduction, indeed. The automata which cannot reproduce on any input will either stuck in some state or fall into endless loops. In any case, some automata will reproduce and give birth to still larger automata. In the worst case this will be automata functionally identical to their parents, having an equivalent transition relation but larger by one tuple. These automata will reproduce on the same input as their parents did. Eventually, a sufficiently large automaton not appearing in the lineage of its predecessors will be generated having a functionally different transition function and reproducing on different inputs than its parent.

In such a way the evolutionary process will continue, generating among other automata also different automata of increasing size, covering increasingly larger part of the space of all autopoietic automata.

□

In the previous theorem we answered positively the question whether there is an autopoietic automaton which under suitable inputs (supplied in a nondeterministic mode) leads to an unbounded evolution producing automata with increasingly complex computational behavior. Now we will pose in a sense a reverse question. Namely, we will ask whether we can decide, for any autopoietic automaton and any infinite input sequence, whether there is an infinite lineage of automata whose members are all descendants of a given automaton on a given input. This is the problem of *sustainable evolution*. We will show a negative answer:

Theorem 4.2 *The problem of sustainable evolution is undecidable.*

Sketch of the proof. Let A be an autopoietic automaton and let $\mathcal{S}$ be a potentially infinite sequence of inputs. In accordance with the results from Section 3 for each lineage starting by A there is a nondeterministic interactive Turing machine simulating that lineage on $\mathcal{S}$. Now it is obvious that the sustainability problem can be transformed to the halting problem which is undecidable.

□

It is interesting to compare the two previous results. While the first one assures that there is an autopoietic automaton which, when "fed" by proper inputs, will give rise to an unbounded evolution, the second result points to the fact that in general we cannot decide whether an autopoietic automaton will give birth to an infinite lineage of offsprings, under the given input. Thus, sustainability seems to require either adaptation of machines to their environment, or changes in the environment enabling the machines to survive, or both.

5 Conclusions

We have presented a novel model of self–reproducing automata based on the notion of finite state transducers. This model allows studies of the algorithmic variability of information controlling the computational behavior and replication of automata. This is achieved by enhancing the functional abilities of a standard transducer by allowing it to read its own transition relation and based on it, to generate a transition relation of its offsprings. The transition relation of an autopoietic automaton

contains programs both for automaton's information processing tasks and for controlling its own evolution (via its offsprings). This idea allows a fresh look at the mechanisms of variability and inheritance of the "genetic information" passed from the parent to its offspring. Namely, in our model the driving force behind the evolution is an algorithmic procedure which itself can become a subject of an evolution driven by itself, so to say. In contrast to many previous approaches and speculations the evolution in our model is not based primarily on random mutations of randomly chosen parts of the controlling code, but on mutations which are algorithmically directed to those parts of the code which can bring only syntactically correct changes in programs controlling both the computational and evolutionary activities of the automaton at hand. The results showing the equivalency with the interactive nondeterministic Turing machines (Corollary 3.1) point to the great computing power of the lineages of autopoietic automata. There exist autopoietic automata which under suitable input conditions could give rise to unbounded complexity growth along the lineages of offsprings of such automata (Theorem 4.1). This offers a positive answer to the related open problem in the domain of artificial life. On the other hand, Theorem 4.2 shows the fragility of such phenomena — in general one cannot decide whether a lineage will evolve infinitely under given input conditions. The potential of our model in artificial life modelling is the subject of author's ongoing research.

Bibliography

[1] McMullin, B.: John von Neumann and the Evolutionary Growth of Complexity: Looking Backwards, Looking Forwards... Artificial Life, Vol 6. Issue 4, Fall 2000, pp. 347-361

[2] McMullin, B.: Thirty Years of Computational Autopoiesis: A Review. Artificial Life, Vol. 10, Issue 3, Summer 2004, pp. 277 - 296

[3] van Leeuwen, J., Wiedermann, J: The Turing machine paradigm in contemporary computing, in: B. Enquist and W. Schmidt (Eds.), *Mathematics unlimited - 2001 and beyond*, Springer-Verlag, 2001, pp. 1139-1155.

[4] van Leeuwen, J., Wiedermann, J.: Beyond the Turing limit: evolving interactive systems, in: L. Pacholski, P. Ružička (Eds.), *SOFSEM'01: Theory and Practice of Informatics*, 28th Conference on Current Trends in Theory and Practice of Informatics, Lecture Notes in Computer Science Vol. 2234, Springer-Verlag, Berlin, 2001, pp. 90–109.

[5] von Neumann, J.: Theory of Selfreproducing Automata. A. Burks (Ed.), University of Illinois Press, Urbana and London, 1966