Int J Software Informatics, Volume 7, Issue 4 (2013), pp.605–613 Tel: +86-10-62661040
International Journal of Software and Informatics, ISSN 1673-7288 http://www.ijsi.org

Email: ijsi@iscas.ac.cn, ijsi2007@gmail.com

Deterministic Autopoietic Automata

Martin Fürer

(Department of Computer Science and Engineering, Pennsylvania State University,
University Park, PA 16802, USA)

Abstract This paper studies some issues related to autopoietic automata, a model of evolving interactive systems, where the automata produce other automata of the same kind. It is shown how they relate to interactive Turing machines. All results by Jiří Wiedermann on nondeterministic autopoietic automata are extended to deterministic computations. In particular, nondeterminism is not needed for a single autopoietic automaton to generate all autopoietic automata.
Key words: autopoietic automata; evolving interactive systems; interactive Turing machines

Fürer M. Deterministic autopoietic automata. *Int J Software Informatics*, **Vol.7, No.4 (2013): 605–613. http://www.ijsi.org/1673-7288/7/i174.htm**

1 Introduction

In 2001, van Leeuwen and Wiedermann[1] defined evolving interactive systems, in particular sequences of interactive finite state machines with global states, to model infinite computations on an ever changing machine or system of machines. The focus here is not on a single machine solving an instance of a problem and stopping. Instead, the objective is to model a modern system that potentially runs forever over an unbounded number of software and hardware changes. Evolving automata have also been called lineage of automata[3]. For more background information, see Refs. [4, 6].

All results of this paper are related to the paper by Wiedermann[5] studying autopoietic automata, a special kind of offspring-producing evolving machines. The offspring relation defines trees of autopoietic automata. Attention is often focused on a *lineage of autopoietic automata*, a path in a tree of autopoietic automata.

Finite autopoietic automata as defined in Ref. [5] are (nondeterministic) finite automata augmented with the following special features. They have two modes and correspondingly two input options and two output options. The two modes are the reproducing mode (defined by a subset R of states) and the transducer mode (defined by the complementary set $Q - R$ of states.

In reproducing mode the automaton uses a finite read-only input tape and a one-way output tape. The finite automaton operates like a Turing machine. It is a finite automaton though, because the 2-way read-only input tape is of fixed length and there are no additional work tapes.

This research is supported in part by the National Science Foundation under Grants CCF-0728921 and CCF-0964655. Part of this work was done while visiting the University of Zürich.
Corresponding author: Martin Fürer, Email: furer@cse.psu.edu
Received 2009-10-00; Accepted 2011-01-03.

In transducer mode, the automaton does not change the tapes, but reads from an infinite input stream of which it can access one symbol of Σ at a time from an input port, and it writes one symbol of Σ at a time into an output port, producing an output stream.

In either mode, the input tape of the automaton A actually contains the code of A, a straightforward description of the transition relation δ. The code is a sequence of 5-tuples in arbitrary order. Each 5-tuple consists of an observed symbol (on the input tape or in the input port, depending on the mode), a current state, a new symbol or the empty symbol (to be written onto the tape or into the output port, depending on the mode), a new state, and a direction (to move on the input tape in reproducing mode, or a dummy indication of no move in transducer mode). For nondeterministic automata, δ is an arbitrary relation. We use deterministic automata here, meaning that δ is a partial function of the first two components (i.e., the observed symbol and the current state).

Whenever the reproducing mode finishes (by entering a special state $q_1 \in R$), the automaton A splits into 2 automata. One of them is the old A with the same input tape, but with an empty output tape. The other one is the offspring A', using the previous output tape as its input tape, while its new output tape is empty. Both automata start in the start state $q_0 \in Q - R$ with either head at the left end on the respective input tape. Depending on the application, the offspring automaton A' keeps reading from the original input stream continuing at the current position (in Theorem 3.1 and Theorem 3.2 below) or both automata receive new input streams (in Theorem 3.3) as in the corresponding situations in Ref. [5]. At this time, the offspring automaton should have a proper encoding of a new transition function on its input tape, otherwise it stops working.

Naturally, the special state $q_1 \in R$ could be avoided in the definition of autopoietic automata. The machine could jump directly to the start state $q_0 \in Q - R$ in transducer mode. In any case, the effect is that no information can be carried over directly from reproducing mode to transducer mode, as the latter always starts with the same state q_0. This contrasts the opposite transition from transducer mode to reproducing mode, where the initial state in reproducing mode depends on the previous state in transducer mode.

It is possible that the new automaton A' is able to read from a stream over a larger alphabet Σ'. The reason is the binary encoding of δ on the input tape. It allows a potentially infinite alphabet, as the symbol σ_i is just encoded by i (in unary representation). Naturally, as the input tape has a finite length, at any time only a constant number of symbols are allowed in the input stream.

2 The Autopoietic Automaton

Here, we present the computational model in a more formal manner, more or less as presented in Ref. [5].

Definition. An *autopoietic automaton* is a six-tuple $A = (\Sigma, Q, R, q_0, q_1, \delta)$, where

- Σ with $\{0, 1\} \subseteq \Sigma$, is the finite or infinite *input alphabet*. The symbols of Σ are read from the input port and written to the output port, one symbol at a time.

The symbols of Σ are represented by binary encodings on the finite length input and output tapes.

- Q is the finite or infinite set of *states*.
- $R \subset Q$ with $Q \neq \emptyset$ is the distinguished finite set of *reproducing states*.
- $q_0 \in Q - R$ is the *initial state* in which the computation of A starts with the input and output heads on the leftmost cell of their respective tape.
- $q_1 \in R$ is the *final reproducing state*. Entering q_1 finishes a reproducing mode of A and starts a transducer stage of A and A's direct offspring A'.

  The offspring A' starts with the previous output tape of A as its input tape and with an empty output tape. A continues with its previous input tape as its current input tape and an empty output tape. Both machines use the transition relation δ encoded by their current input tape. They start with start state q_0 with the head positioned on the leftmost input tape. In general, both machine receive their own new input stream through their input port.

  When focusing on a lineage, we can assume that there is just one input stream and the successor machine keeps reading where the predecessor machine has stopped, because each autopoietic automaton in the lineage only reads a finite string.
- $\delta \subseteq \Sigma \cup \{\epsilon\} \times Q \times (\Sigma \cup \{\epsilon\}) \times Q \times D$ is the *transition relation*. The transition relation determines the possible actions for any given symbol and state. An action consist of a new symbol, a new state and a direction. In transducer mode, the given symbol is read from the input port and the new symbol is written onto the output port. In reproducing mode, the given symbol is read from the input tape, and the new symbol is added at the right end of the output tape. The head on the input tape is moved in the given direction of D. Possible values are d_0 (left), d_1 (no move), and d_2 (right). A further value d_3 of D is not a real direction, but indicates that the current transition belongs to the transducer mode. The value ϵ represents an empty word indicating that no symbol is read or written.

  For deterministic machines, the transition relation is a *partial transition function* from $\Sigma \cup \{\epsilon\} \times Q$ to $(\Sigma \cup \{\epsilon\}) \times Q \times D$. Furthermore, if $\delta(\epsilon, q)$ is defined, then $\delta(\sigma, q)$ is undefined for every $\sigma \in \Sigma$.
- The inscription of the input tape at any time is a sequence (in any order) of encodings of the transitions of δ. The transition $(\sigma_i, q_j, \sigma_k, q_\ell, d_m)$ is encoded as $10^{i+1}10^{j+1}10^{k+1}10^{\ell+1}10^{m+1}1$. The empty word is encoded by the empty string.
- We might require states of R to have odd index and states of $Q - R$ to have even index. Then j being odd would require ℓ to be odd too.

We also use the notion of an *interactive Turing machine*, as defined by Refs. [1, 2]. This is a nondeterministic Turing machine with one work tape, reading its input a symbol at a time from an input port connected to an input stream and writing its output a symbol at a time onto an output port creating an output stream. Thus the

input and output mechanisms are the same as for an autopoietic automaton. Hence, the differences to typical Turing machine definitions are the following.

- Not only is the output produced from left to right (usually referred to as having a one-way output tape), but also the input is read from left to right (whereas typically, on a read-only input tape the head can move arbitrarily).
- The input and output streams are of infinite lengths.
- There is just one tape. Typically, in addition to the input and output tapes (which are here replaced by input and output streams), some finite number of work tapes is allowed.
- Interactive Turing machines are defined (by default) as nondeterministic.

Here, we actually use *deterministic interactive Turing machines*. They are defined exactly as (nondeterministic) interactive Turing machines, except that they are deterministic, i.e., the transition relation δ is a partial function defined on $\Sigma \times Q$.

3 The Theorems

Recall that a lineage $\mathcal{A} = A_1, A_2, \ldots$ of autopoietic automata is a path in the tree defined by the offspring of a single autopoietic automaton A_1. When talking about a lineage of automata, we require A_{i+1} to be a new offspring of A_i rather than the replica of A_i for every i. We also assume that there is just one input stream. The offspring automaton keeps reading from the position reached by the parent, even though the input stream contains symbols from larger and larger alphabets as it reaches parts intended for later automata A_i. Thus we have just one such essential lineage for every input stream, as our automata are deterministic.

Later, we will also consider arbitrary trees obtained even in the deterministic case, by not focusing on one lineage and considering new input streams after each branching.

The following two theorems are proved exactly as in the original version[5] where both the automaton and the Turing machine are nondeterministic.

Theorem 3.1. A lineage $\mathcal{A} = A_1, A_2, \ldots$ of deterministic autopoietic automata can be simulated by a deterministic interactive Turing machine.

Proof.

The interactive Turing machine can easily maintain copies of the input and output tapes of the simulated autopoietic automaton at all times. Thus it can always consult the copy of the input tape to determine the next simulation step.

Naturally, the Turing machine has fixed input and output alphabets. Therefore, for the theorem to hold, we have to use an appropriate notion of simulation. The interactive Turing machine receives an input stream that consists of a sequence of binary encodings of the symbols on the input stream of the simulated autopoietic automaton. Likewise it produces an output stream encoding the output stream of the simulated automaton. Simulation means step by step modeling of the computation of the autopoietic automaton by using a special encoding to represent the current state on a work tape.

Further details of the proof can be copied from the nondeterministic version[5].

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

Proof.

Without loss of generality, we assume that M has just one tape (infinite to the right only) and uses the alphabet $\Sigma_T = \{0, 1, b\}$, since such a machine has the same computational power as any multi-tape machine. The automaton A_i handles the simulation as long as the Turing machine M only uses i tape cells. A state of A_i not only encodes the corresponding state of M, but also the tape inscription (of fixed length i) and the head position of M.

Whenever the Turing machine uses a new tape cell, the simulating automaton switches to reproducing mode. It copies the part of the automaton involving the states of R (handling reproduction). The part involving the states of $Q - R$ (handling transducer steps) is roughly tripled in length, corresponding to the additional tape cell containing 0, 1, or b (blank). Also the few additional transitions corresponding to the head being on the new cell are easily handled.

As the details are tedious, we illustrate the functioning of this simulating lineage of autopoietic automata by an example. Assume, the simulated interactive Turing machine seeing an input symbol $\sigma \in \Sigma$ and a tape symbol $\tau \in \Sigma_T$ in state q, outputs σ', writes τ' on the work tape, goes to state q', and moves the head on the work tape to the right. Then for every $yz \in \Sigma_T^{i-1}$, there are corresponding states $\mathrm{code}(y, q, \tau, z)$ (indicating a tape inscription $y\tau z$ with the head observing the symbol τ located between y and z) and transitions $\delta(\sigma, \mathrm{code}(y, q, \tau, z)) = (\sigma', \mathrm{code}(y, \tau', q', z), d_2)$ of the simulating automaton A_i.

In reproducing mode the corresponding 5-tuple $(\sigma, \mathrm{code}(y, q, \tau, z), \sigma', \mathrm{code}(y, \tau', q', z), d_2)$ has to be replaced by the three 5-tuples $(\sigma, \mathrm{code}(y, q, \tau, z\theta), \sigma', \mathrm{code}(y, \tau', q', z\theta), d_2)$ for $\theta \in \Sigma_T = \{0, 1, b\}$. In addition commands for handling an observed symbol at position $i+1$ have to be produced. Furthermore, commands for interrupting the simulation have to be modified by replacing the states $\mathrm{code}(x, q)$ with $x \in \Sigma_T^i$ by the three options, $\mathrm{code}(x0, q)$, $\mathrm{code}(x1, q)$, and $\mathrm{code}(xb, q)$.

Again, some additional details can be found in the proof for the nondeterministic version[5].

One important step has been forgotten in Ref. [5]. It could be avoided by omitting the unmotivated restriction in the definition of autopoietic automata requiring them to always start the transducer mode with the same state $q_0 \in Q - R$. With this restriction, the simulation of the interactive Turing machine using $i + 1$ tape cells by A_{i+1} cannot just continue where the simulation of the interactive Turing machine using i tape cells by A_i has stopped, because the state of the interactive Turing machine has been lost.

This problem can be solved though, because the simulated state q can be carried over to the reproducing phase. Then the first thing is to write a transition on the output tape requiring to jump from state q_0 to q without reading or writing anything. □

More interesting is the next theorem, again corresponding to the following nondeterministic version in Ref. [5]. There exists an autopoietic automaton which, when working in nondeterministic input mode, generates a descendant tree containing all autopoietic automata. Here, "working in nondeterministic input

mode" refers to allowing various inputs to create various children of a single autopoietic automaton.

Naturally, one cannot produce all autopoietic automata on one path, because many automata (on some or all input streams) never switch to reproducing mode. Thus, whenever any such automaton is produced, the reproduction stops on that path.

In the nondeterministic case, there is a trivial proof using the full power of nondeterminism of an autopoietic automaton to write anything on its output tape during reproducing mode. Wiedermann[5] does not explain why he chooses a more explicit construction. In our case, using deterministic autopoietic automata, the simple construction is not an option, and we have to be much more explicit.

At a first glance, one might want to impose the nice property that all the automata of the tree only write legal encodings of autopoietic automata onto their output tape. In particular, one would want to disallow writing an infinite output sequence. But naturally, such strong requirements cannot be enforced. If we want to produce all autopoietic automata, we have also to produce those that write various kinds of garbage onto their output tape, including any computable infinite output sequence.

What can be obtained is a tree T of autopoietic automata with the following properties.

- There is an easily identifiable subtree L of the vertices of T such that every autopoietic automaton is represented by a child of a node of L.
- All the children of nodes of L represent autopoietic automata.
- All the nodes in L are well behaved automata, meaning that for all input sequences, they only write proper encodings of autopoietic automata onto their output tapes.
- In particular, the automata in L finish their reproducing mode after finitely many steps.

Without claiming such properties, Wiedermann's tree of automata has these properties and ours will have them too. In addition, we would like our automata in L to be deterministic. But this seems too difficult a goal.

We slightly stretch the definition of a deterministic automaton by requiring the partial function property of the transition relation δ not for all states, but only for those reachable from the start state. An automaton is *essentially deterministic* if from all reachable configurations at most one move is possible.

The proof for essentially deterministic automata is much more difficult than the proof for nondeterministic automata, because the various autopoietic automata have to be created in a more systematic way. We cannot use nondeterminism to help producing them. Still, even in our deterministic model, a whole tree of automata is created depending on the input sequences. Thus, nondeterminism is still present in some sense through the variation of the input stream. As mentioned above, we refer to this as working in nondeterministic input mode[5].

But during the reproducing mode, the input stream is not touched, allowing just one new automaton to be produced during any reproducing phase of a deterministic

autopoietic automaton. Nevertheless, different inputs allow to start the reproducing phase in different states and thus producing different offspring.

A fundamental requirement of autopoietic automata is that they have to act according to the program stored on their input tape. Actually, the proof of the nondeterministic case[5] does not explicitly say how this task is handled. But in that version, nondeterminism (used during the reproducing mode) is very helpful for the creation of any possible autopoietic automaton as an offspring.

Theorem 3.3. There exists a tree T of autopoietic automata, where all autopoietic automata occur as children of the nodes of a subtree L consisting of essentially deterministic autopoietic automata.

Proof.

We describe a type of essentially deterministic autopoietic automaton A, producing in a systematic way all autopoietic automata. Hereby, the automata of type A itself evolve over time, while always exhibiting roughly the same functionality. More precisely, the reachable part of all automata in L are identical (of type A). In a systematic way, non reachable parts are added in order to make them reachable later when leaving L.

Every autopoietic automaton acts according to its transition function δ encoded by a sequence of 5-tuples on its input tape. We want to slowly produce new automata without yet affecting their operation. For this purpose, we partition the states of A into active and passive states. Both sets are also partitioned according to transducer mode and reproducing mode.

The active states govern the operation of A, while the passive states have no such effect. This can be achieved, by simply making all the passive states inaccessible from the start state. The 5-tuples representing the transitions of δ with active states are stored in a block after the block of 5-tuples involving the passive states. The beginning of the active block is marked by the first transition involving the start state q_0.

The passive states are q_3 up to some q_k, where k grows over time. The active states are q_0, q_1 and a finite set of states with indices greater than k. Only the active states are reachable from q_0. By keeping the indices of the active states hight, they don't interfere with a systematic production of the passive part.

The systematic composition of new automata stops upon reading a symbol σ_0 from the input port. At that time, the automaton switches to reproducing mode in order to create the automaton encoded by the passive states. For that purpose, the producing automaton basically copies the passive block to the output tape, but it decreases every state index by 2. It does not copy the active block, thus stopping the systematic production of automata. Instead, it just enters state q_1 to finish its reproduction mode.

When reading any symbol other than σ_0 the producing autopoietic automaton enters reproducing mode to continue the systematic production of all possible passive parts in order to create all automata. The passive part is copied to the output tape with some minor adjustment. The active part is made more powerful as described below and all the indices of active states, except 0 and 1 are increased by 1 to create space for a new passive state.

The minor adjustments to the passive part are as follows. When reading σ_1, then

the new passive instruction $(\sigma_0, q_2, \sigma_0, q_2, 0)$ is written onto the output tape, before the other instructions are copied. When reading σ_{j+2} for $j \geqslant 0$, then the j-th index in the instruction sequence is increased by 1 (or cycled through its finitely many possible values), while otherwise, the passive part is just copied.

The modification of the active part during the reproduction process is more complicated. If so far, the active part is able to read symbols $\sigma_0, \ldots \sigma_{j+1}$, then the ability to read and properly process the symbol σ_{j+2} is added. A constant number of new instructions containing a fixed number of new states is sufficient to do this goal.

Here are a few more details. The new instructions can be produced, because they are patterned after the previous such addition. The new commands are for reading symbol σ_{j+2}, entering a new state, walking the head on the input tape to the next index, and entering a state that had been added in the previous round. From this state, further head movements are performed, hereby reaching the j-th index in the sequence of 5-tuples (machine code) in order to increase it during the copying process.

Wiedermann[5] has used the same two operations of increasing the number of instructions and increasing some indices in the instruction sequence. His nondeterministic automaton could do such things at any time, while our deterministic automaton needs the flexibility of the input to do such operations at input determined locations. In any case, the deterministic autopoietic automaton A systematically produces all possible autopoietic automata.

The problem of sustainable evolution asks for any autopoietic automaton and any infinite sequence of inputs whether there is an infinite lineage generated by that automaton on that input sequence. To have a precise question, one would have to restrict attention to (in some given system) definable infinite sequences of inputs. But it is even undecidable for a fixed sequence. Except for avoiding the accidentally wrong direction, the result is shown the same way as for nondeterministic automata[5].

Theorem 3.4. The problem of sustainable evolution for deterministic autopoietic automata is undecidable even for the all 0 input stream.

Proof.

By Theorem 3.2, every deterministic interactive Turing machine can be simulated by a lineage of autopoietic automata. This implies that the halting problem for Turing machines can be reduced to the question whether an automaton creates an infinite lineage as follows. A lineage is built that simulates the given Turing machine with the empty input. Every automaton in the lineage creates a single offspring after a constant number of simulation steps. This offspring continues the simulation. The lineage can be defined such that it stops as soon as the simulation stops. This way the lineage is finite if and only if the Turing machine halts.

4 Conclusion

We have shown that autopoietic automata need not be nondeterministic to have the nice properties shown in Ref. [5]. Only the construction of an automaton generating all automata gets significantly more complicated in the deterministic setting.

References

[1] van Leeuwen J, Wiedermann J. Beyond the turing limit: evolving interactive systems. In: Pacholski L, Ruzicka P, eds. SOFSEM 2001: Theory and Practice of Informatics. 28th Conference on Current Trends in Theory and Practice of Informatics Piestany. Slovak Republic. Nov. 24–Dec. 1, 2001. LNCS 2234, Springer. 2001. 90–109. http://link.springer.de/link/service/series/0558/bibs/2234/22340090.htm.

[2] van Leeuwen J, Wiedermann J. The turing machine paradigm in contemporary computing. In: Enquist B, Schmidt W, eds. Mathematics Unlimited—2001 and Beyond. Springer-Verlag. 2001. 1139–1155.

[3] Verbaan P, van Leeuwen J, Wiedermann J. Complexity of evolving interactive systems. In: Karhumäki J, Maurer HA, Paun G, Rozenberg G, eds. Theory is Forever, Essays Dedicated to Arto Salomaa on the Occasion of his 70th Birthday. LNCS 3113, Springer. 2004. 268–281. http://springerlink.metapress.com/openurl.asp?genre=article&issn=0302-9743& volume=3113&spage=268.

[4] Verbaan P.R.A. The Computational Complexity of Evolving Systems[Ph.D. thesis]. Proefschrift Universiteit Utrecht. 2006. http://dspace.library.uu.nl/bitstream/handle/1874/7653/full.pdf?sequence=14.

[5] Wiedermann J. Autopoietic automata: Complexity issues in offspring-producing evolving processes. Theor. Comput. Sci. 2007, 383(2-3): 260–269.

[6] Wiedermann J, van Leeuwen J. How we think of computing today. In: Beckmann A, Dimitracopoulos C, Löwe B, eds. Logic and Theory of Algorithms. LNCS 5028. 4th Conference on Computability in Europe. CiE 2008. Athens, Greece. June 15-20, 2008. Springer. 2008. 579–593. http://dx.doi.org/10.1007/978-3-540- 69407-6_61.