Self-Programming of Matter and the Evolution of Proto-Biological Organizations

Steen Rasmussen
Rasmus Feldberg
Carsten Knudsen

SFI WORKING PAPER: 1992-07-035

SANTA FE INSTITUTE

SELF-PROGRAMMING OF MATTER AND THE EVOLUTION OF PROTO-BIOLOGICAL ORGANIZATIONS

Steen Rasmussen[1], Rasmus Feldberg[2], and Carsten Knudsen[2]

[1]Theoretical Division and Center for Nonlinear Studies, MS B258
Los Alamos National Laboratory, Los Alamos NM 87545
USA
[1]Santa Fe Institute, 1660 Old Pecos Trail, Santa Fe NM 87501
USA
[2]Physics Laboratory III and Center for Modeling, Nonlinear Dynamics, and
Irreversible Thermodynamics, B 306,
The Technical University of Denmark, DK-2800 Lyngby
Denmark

To appear in the proceedings of the meeting "Frontiers of Life",
held in Blois, France, October 1991.

We propose a novel class of formal dynamical systems, which are referred to as *self-programming* or *constructive*. We believe that these systems form natural models for functional self-organization in prebiotic and proto-biological systems. We shall use the computer as a "test-tube" to follow the dynamics of these systems and in particular investigate their emergent functional properties. Using the computer as a simulation laboratory is in this context based on two fundamental assumptions:

(i) That the logic, e.g. the dynamics and the functional organization, of a physical and in particular a living system can be described in terms of mathematics and hence can be studied independent of the molecular hardware that carries this dynamics.

(ii) That the properties of structures and organizations emerging from the dynamics of self-programming systems reflect aspects of the organizing principles of prebiotic and proto-biological systems. We shall show that for instance the emergence of complex cooperative structures is part of the natural dynamics of self-programming systems.

The first assumption gives rise to a number of issues of quite fundamental nature, which we shall only touch a little bit in the following [16][17][31][28]. The second assumption rises the question of what kind of dynamical systems, read: mathematical frameworks, can be used to study the development of life-like organizations and structures? We shall argue that members of the class of self-programming universal computers are suitable candidates and give specific examples of what kind of dynamical properties these systems have and how they may relate to the more complex physical systems.

1 The basic concept

Maybe the clearest way to communicate the fundamental idea behind this approach is to look at the dynamics of molecular evolution:

Molecular evolution seems to be characterized by a successive emergence of new functional properties carried by molecules. The physical properties (shape, charge, etc.) of the chemical species together with macroscopic thermodynamical properties, such as temperature and pressure, define their possible interactions with other molecules and thereby their functional properties. Chemical systems create new properties through recombination of molecules via chemical bonds. New combinations between existing molecules and combinations of new molecules with other molecules then define new functional properties in a system at large. We shall in the following refer to this loop: *molecules* → *physical properties* → *functional properties* → *interactions* → *new molecules*, as an example of autonomous programming, or self-programming, of matter. See figure 1.a.

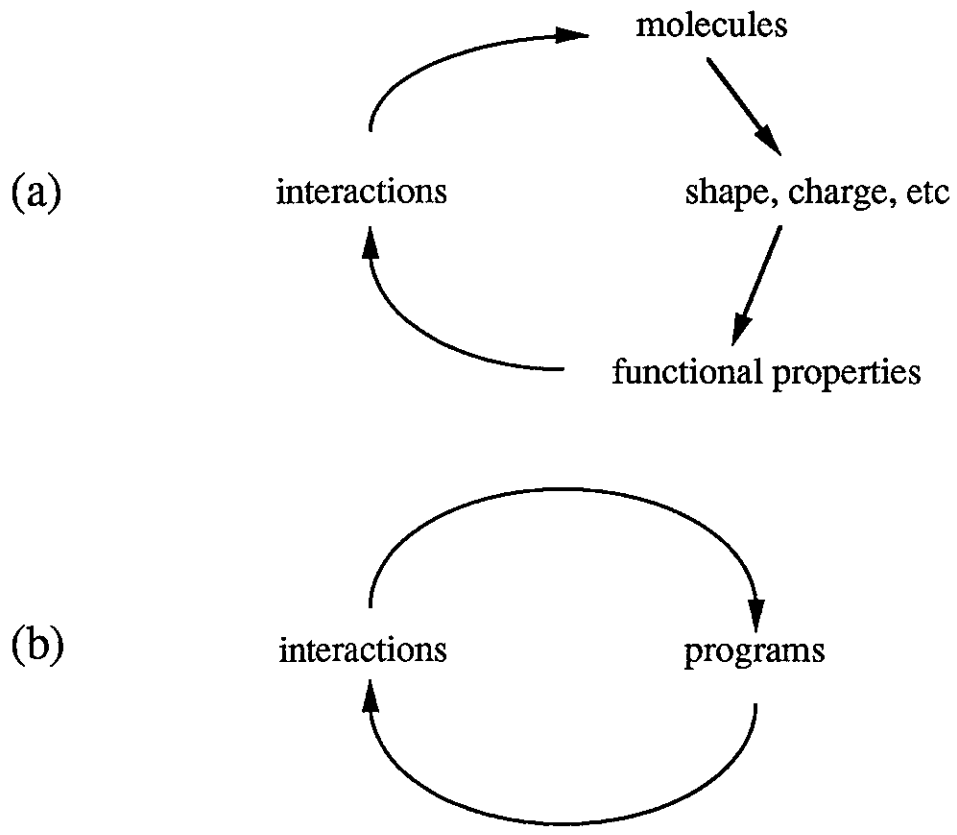

FIGURE 1 (a) Evolution of novel functional properties in chemical systems. This is an example of autonomous programming of matter. (b) The functional self-programming loop implemented on a computational system.

The essence of this chemical loop can be extracted by choosing another kind of pro-

grammable "matter" in which the elements are able to alter each other in a more direct way and recombine to create new functional properties. In other words it is a loop abstracted from molecular reactions and re-implemented in a more accessible media without the particular physical interpretation in chemistry. Clearly, any implementation implies a physical interpretation. However, for a computational implementation we do not need to worry about the physical interpretation. The computer will take care of that. Thereby the programming loop becomes: *functional properties* $\rightarrow$ *interactions* $\rightarrow$ *new functional properties*, as new elements with new functional properties are formed. See figure 1.b. In the computational systems to be discussed such functional elements are equivalent to programs or assemblies of machine code instructions. This general approach is closely related to Fontana's (1991) and McCaskill's (1990).

It is important to note that the elements of these constructive systems primarily have *deterministic* interaction rules (recall chemical reactions). This is in opposition to how novelty normally is viewed to emerge in biological systems, namely through random processes. We shall focus on deterministic self-programming, but also discuss the consequences of random perturbations of self-programming processes.

One can ask why we make these abstractions to study the dynamics of self-programming? Why do we not directly model macro-molecular interactions, since we are interested in the principles of prebiotic evolution? The answer to this question has two parts and the first is very simple: The computational power of current computers does not allow such an approach. The computational power of a few micromoles of reactants in a few microliters of solvent just for a split of a second surpasses with many orders of magnitude all computational power man is in control over on the planet. It will take a long time before computers will allow this kind of studies. There is, however, also another reason for performing this exercise in abstraction: If we believe that life did not emerge as an accident, e.g. as a highly improbable event, but rather as a consequence of self-organizing processes. Then a study of the dynamics of programmable matter may shed light on the nature of these self-organizing processes, since these constructive systems are "designed" to produce new functional properties.

2 Computation

Already von Neumann noted that programs in computers are identical to data, programs alter data, and thus programs can alter programs [34]. This is what makes self-programming possible with computers.

There exist several well established theoretical frameworks, the different computation paradigms, which in principle can describe such self-programming processes. Any computational system from the class of universal formalisms – including Turing machines [33][21][12], cellular automata [30][18], the lambda calculus [2][4], Post systems [24][21], general recursive functions [15], classifier systems [11][10], partial differential equations [23], hard billiard ball computers [20], von Neumann machines [34][12], and even smooth maps on the unit square [22] – can in principle be turned into a self-programming system. However, some formalisms seem to have a more natural way of being self-programming, in the sense we discussed in the previous section, than others. It is also worth noticing that the autonomous dynamics

1 The basic concept

Maybe the clearest way to communicate the fundamental idea behind this approach is to look at the dynamics of molecular evolution:

Molecular evolution seems to be characterized by a successive emergence of new functional properties carried by molecules. The physical properties (shape, charge, etc.) of the chemical species together with macroscopic thermodynamical properties, such as temperature and pressure, define their possible interactions with other molecules and thereby their functional properties. Chemical systems create new properties through recombination of molecules via chemical bonds. New combinations between existing molecules and combinations of new molecules with other molecules then define new functional properties in a system at large. We shall in the following refer to this loop: *molecules* → *physical properties* → *functional properties* → *interactions* → *new molecules*, as an example of autonomous programming, or self-programming, of matter. See figure 1.a.

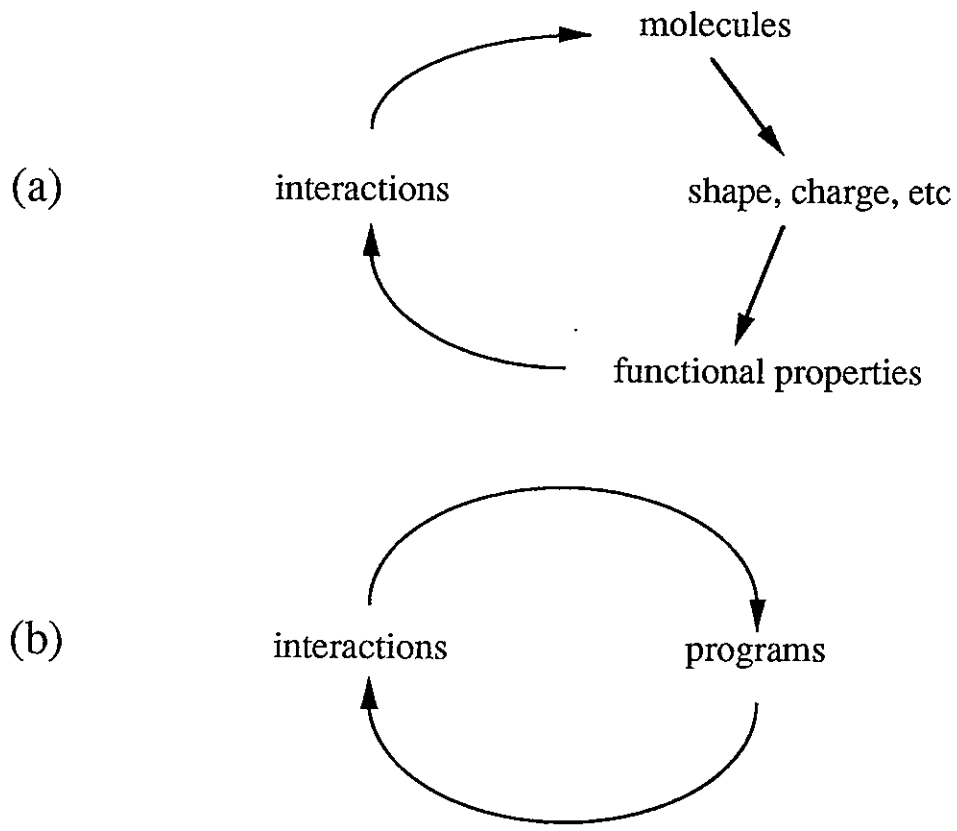

FIGURE 1 (a) Evolution of novel functional properties in chemical systems. This is an example of autonomous programming of matter. (b) The functional self-programming loop implemented on a computational system.

The essence of this chemical loop can be extracted by choosing another kind of pro-

of computational systems has never been the subject for any serious investigations.

A universal machine can simulate the information transformation rules of any physical mechanism for which those rules are known. In particular, it can simulate the operation of any other universal machine – which means that all universal machines are equivalent in their simulation capabilities. It is an unproven, but so far uncontradicted, contention of computation theory that no mechanism is more powerful than a universal Turing machine. This conjecture is known as the *Church-Turing thesis* [12].

Our physics is at least able to support universal computation, since we are able to build digital computers. However, we cannot be sure whether or not Nature is able to support a more general class of information processing. The claim that *any* physical process can be precisely described by a universal machine, is often referred to as the *physical Church-Turing thesis* [9].

Technically, the common property for all universal formalisms is that their range is equivalent to the class of partial recursive functions (computable functions $\equiv$ functions which can be specified through an effective procedure). A less formal way of identifying the class of computable functions is to identify them as functions which are "human computable". Not all computations can be performed in a finite number of steps, since it can be shown that some computable functions will never let a computation halt. Some, or all, values of such functions are therefore not defined. There exists no algorithm by which we can decide in the general case whether a random chosen function would let a computation halt or not. This is referred to as the *halting problem* [21]. This implies that the "trajectory" of a randomly chosen initial condition for a computation universal system cannot be know *a priory*. This also implies that dynamical open-endedness in principle exists in such systems, which is a property we want.

3 An example

Note that the self-programming dynamical system we are going to investigate is not a model of a particular physical, chemical, or biological system. We are interested in the generic properties of the dynamics of programmable matter. By implementing the property of self-programming in a simple computational system we hope to be able to extract such generic properties. Hopefully the dynamics of the emergence of novel properties in such a simple system also enables us to understand aspects of the emergence of novel properties in the more complex natural systems.

The first thing to do is to pick a computational paradigm and thereafter alter it to allow it to program itself; to make it self-referential.

Now follows a short description of how the dynamics of self-programming in simple von Neumann like systems are investigated. Note that this system is not implemented directly in the machine code of the computer being used, but a von Neumann like machine is being *simulated* on the host computer. We do that partly to have a more flexible system and partly out of security reasons. In this way no autonomous self-programming processes can "escape" from the computer onto the network and make damage. These processes could potentially be dangerous, since they have a potential to evolve and thus adapt to whatever environment

they end up in. Fortunately we have not yet encountered evolving computer viruses on the network [32].

The details of a multitasking von Neumann architecture, which is the architecture most modern computers are based (IBM, Macintosh, SUN, CRAY, to mention a few), is quite complicated, so we cannot get too technical here. Almost any introductory textbook on the theory of computation has a description of the von Neumann computer (see for instance [12]) and a throughout discussion of the technical details relating to how to make computers self-programming can be found in [27].

The self-programming system consists of a one dimensional computer memory of size N with periodic boundary conditions where each memory slot contains one machine code instruction out of a set which form a universal set. The details of the instruction set is not important for the following. We just assume that it is universal, which means that any function can be constructed by combining members from the set. The instruction set includes an instruction (SPL) which will start a new process (also called a program pointer) at a specified location in the memory when executed. Thereby new processes can be created. Processes can be eliminated either when they meet a certain instruction or when the finite execution queue of size L is filled. All the machine code instructions are of the form

$$\mathrm{INS}(A, B), \tag{1}$$

where A and B are positive or negative integers and where the function of the instruction INS can be the one discussed above or for instance: Move the instruction which is A memory slots away into the memory slot which is B memory slots away (MOV). The instruction can also be of the form: Continue the current process at the memory location A slots away if the second argument of the instruction at this location is positive (JMP). A notion of locality in the memory can be imposed, by only allowing local read and write privileges (A and B cannot exceed a certain numerical value) and/or through a local address specific definition of computational resources. A variety of other properties like updating schemes for the processes can also be regulated. For details, consult Rasmussen et al. (1990 and 1991) and Knudsen et al. (1991).

It should be noted that it is a non-trivial issue to define observables in these dynamical systems and that the analysis of their dynamics is in general quite complicated. We shall, however, not get into these details here.

For a broad range of system parameters a randomly initialized memory of machine code instructions with a random distribution of processes, or program pointers, the system has an ability to develop cooperative programs which eventually take over the whole computer memory.

A typical example of such a dynamics is the evolution from a randomized memory into the development of the ordered and stable, so called, MOV-SPL structure, which consist almost entirely of MOV and SPL instructions. The evolution of the system goes through a number of epochs where other instructions and organizational structures are dominating before the MOV-SPL structure eventually takes over. This particular evolutionary path depends on the details of the initial randomized memory. Under other (random) initializations we experience that the system develops different structures and in other cases it dies. This variety of evolutionary pathways occurs with purely deterministic rules of interaction.

Adding a little noise to the process, e.g. making some of the instructions a little sloppy in their functions, does not seem to have a major effect on the quality of the evolutionary dynamics.

The viability of the MOV-SPL structure stems from a cooperation between the MOV and the SPL instructions with appropriate arguments. The MOV instruction with very high probability either copies another MOV instruction or a SPL instruction, guaranteeing the reproduction of the structure. The SPL instruction hands out pointers (or processes) either to another SPL instruction or to a MOV instruction, thereby guaranteeing that the structure is kept alive.

Note that this cooperative structure is a *spatio-temporal* structure which reproduce as *a whole* and not through an individual reproduction of the each of the parts. The important steps closing the inter-connected cooperative cycles in the "interaction graph" for the MOV-SPL structure are given in equations (2) - (4). S corresponds to SPL, M corresponds to MOV, $(*)$ corresponds to an excess pointer created by a SPL instruction, and $(\cdot)^{(*)}$ corresponds to an instruction with a pointer.

$$S^{(*)} \rightarrow S^{(*)} + (*)\,, \tag{2}$$

$$S^{(*)} + M' + M + S' \rightarrow S + M' + M^{(*)} + S' + (*) \rightarrow S + 2M' + M + S^{(*)'} + (*)\,, \tag{3}$$

and

$$S^{(*)} + S' + M + S'' \rightarrow S + S' + M^{(*)} + S'' + (*) \rightarrow S + 2S' + M + S^{(*)''} + (*). \tag{4}$$

Another structure frequently found in this system is made up by other instructions - mainly MOV- and JMP instructions. The important steps closing the cooperative loops for this structure are given in equations (5) - (7). J corresponds to JMP.

$$J^{(*)} \rightarrow J^{(*)}\,, J^{(*)} \rightarrow M^{(*)}\,, \tag{5}$$

$$M^{(*)} + J + J' \rightarrow M + J^{(*)} + 2J'\,, \tag{6}$$

and

$$M^{(*)} + J + M' \rightarrow M + J^{(*)} + 2M'. \tag{7}$$

The functional dynamics defined through these equations are extracted from the two interaction graphs and it is seen that invariance or recurrence in the iterated interaction graph is equivalent to functional cooperation. Invariance is trivially fulfilled when the system is on an attractor, but recurrence frequently occurs when the system is on a transient and the recurrent states can thereby define metastable epochs where the systems can be for long periods of time.

In is interesting to note that these cooperative structures can be highly perturbation stable. This can be tested by adding noise to the systems either during their evolution and/or by perturbing the structures by putting other programs into the systems.

It is also interesting to note that the autocatalytic sets found in systems of catalyzed cleavage-ligation reactions of polymers (Kauffman (1986), Farmer et al. (1986), and Bagley et al. (1991)) also reproduce or maintain as a whole and not through an individual reproduction of each of the parts. Such autocatalytic sets seem to be a special case of a more general form of self-referential attractor structures found in self-programming systems.

There is an extensive literature on cooperation in different types of dynamical systems. For a discussion on the nature of cooperative structures in prebiotic systems see for instance Eigen (1971), Eigen and Schuster (1979), Kauffman (1986), Farmer et al. (1986), Bollobás and Rasmussen (1989), Rasmussen (1989), Bagley and Farmer (1991), Fontana (1991), or Rasmussen et al. (1991).

4 Generic properties

Obviously one should think that the details of the dynamics of a self-programming system depends on the details of the computational representation. This is also well established through a number of experiments with these systems (Fontana (1991), Rasmussen et al. (1991), and Ray (1991)).

However, it is also clear that certain general representation independent principles apply to the dynamics of such systems. For instance, the notion of *an attractor* (in dynamical systems terms) for a self-programming system has as somewhat different meaning than an attractor in for instance a differential equation system (for the technical details, see [27]). In self-programming systems, function carrying entities (functions/programs/molecules), interact with themselves and other entities (functions/programs/molecules) and when an attractor is reached the whole set of interacting entities (functions/programs/molecules) are maintained or being "reproduced" through the system dynamics. These entities define a closed organizational form as seen in the above examples of the MOV-SPL and the JMP-MOV structures. One can interpret the dynamics as being a cooperation between the involved entities.

5 Artificial and natural self-programming systems

We have extracted the property of self-programmability which characterize the processes producing novelty in molecular systems. We have implemented this property in much simpler and more tractable dynamical systems. By characterizing the dynamics of these simple self-programmable systems we have tried to characterize dynamical aspects of the emergence of novel properties for evolutionary processes which may also be of importance for prebiotic and proto-biological systems.

This step away from the use of "classical" dynamical systems, e.g. systems where the equations, or rules of dynamics, do not depend on the dynamics (typically ordinary differential equations), has opened a new alley. The best current models for truly evolutionary systems seem to be the different computational paradigms and related formalisms.

We have created a particular kind of self-programmable matter based on a modified von Neumann machine. We have shown that the functional dynamics can be defined through

an interaction graph and that invariance or recurrence in the iterated interaction graph is equivalent to functional cooperation. Invariance is trivially fulfilled when the system is on an attractor, but recurrence frequently occurs when the system is on a transient and the recurrent states can thereby define metastable epochs.

Frozen accidents in terms of particular functional properties are a natural consequence of runaway processes accompanying self-organizing phenomena in these systems. However, complex cooperative dynamics is not guaranteed even given the right conditions. Qualitatively different cooperative structures can emerge from systems with the same parameters and the system can even die as a result of the evolution. Chance is part of the game in these systems either through the definition of the initial conditions or through a slightly noisy dynamics. This indicates that many properties characterizing contemporary life-forms should be frozen accidents.

Functional stability to perturbations is a *product* of evolution and not a property of the details of the underlying programmable matter. Even when the system is based on a von Neumann architecture, which is highly perturbation fragile with respect to functional properties, it can spontaneously develop perturbation stable cooperative structures. Functional stability to perturbations as we see it in modern life-forms should therefore be a product of evolution and not a property of the underlying chemistry. This is probably true, since an embedding of any modern biochemical pathway in a random chemical environment will cause the pathway to collapse.

Cooperation emerges as a natural property of the functional dynamics in systems with a constructive dynamics. The intricate network of functional cooperation characterizing contemporary life should therefore be a natural consequence of the dynamics in the prebiotic environments, and not the result of a long chain of highly improbable events.

A general feature for most of the emerging cooperative structures is a reproduction of the structure *as a whole* and not an individual reproduction of the parts. It seems easier to create a reproductive system without separate self-replicating "genes". In the above systems, the interaction graph is part of the "template information". Maybe such organizations preceded the modern biological organizations.

Finally we want to stress that an organism and its environment form an *integrated* system. The more low-level the (proto-) living process is, the more fuzzy the organism-environment distinction appears. At the level where we are operating with the self-programming systems such a distinction is clearly arbitrarily.

6 Summary

Macroscopic, spatio-temporal, cooperative structures spontaneously emerge in these self-programming system. They emerge in a similar way as macroscopic dissipative structures do in physico-chemical systems. A thermodynamical interpretation of the computational system yields an equivalence between the flux of free energy and the flux of computational resources (executions per system iteration), an equivalence between computer memory and space, and an equivalence between the microscopic degrees of freedom in the physico-chemical system and all the possible functional interactions in the computational system. A notable

difference is, however, that our macroscopic computational structures *change* even with a constant pumping (allowed executions per iteration). Due to our system's self-programmable properties it does not stay in a fixed macroscopic pattern, as for instance the Raleigh-Bénard convection or the chemical reaction waves in the Belousov-Zhabotinski reaction do for constant pumping. Our systems have in addition a property biological systems also have: they can change themselves and thereby undergo a development.

The details of the emerging structures depend on the details of the underlying systems as demonstrated through variations in the instruction set, the updating scheme, the parameters, and the initial conditions of the systems [26, 14, 27]. The functional details of biomolecular based proto-organisms are therefore also assumed to depend on system details. Although the above investigations do not explain how matter has been able to organize into the complex structures we see in contemporary biological organisms, we have seen that even simple self-programmable systems are able to develop complex adaptive organizations. These structures are still far simpler than, and probably also organized differently from, any contemporary biological organism. Nevertheless the structures are able to channel and focus the available computational resources very effectively, they are perturbation stable, they are able to maintain themselves (e.g. reproduce), they can die, and they can undergo evolution. These properties enable the structures to expand and invade the whole system for long periods of time. Functional cooperation is an inherent property of the dynamics of programmable matter.

References

[1] Bagley R. and D. Farmer, Spontaneous emergence of a metabolism, in Artificial Life II, SFI Studies in the Sciences of Complexity, Vol. X, eds. Langton, C. et al., Addison-Wesley, Redwood City, 1991, p 93-140.

[2] Barendregt H., The lambda calculus. Its syntax and semantics, Studies in Logic and the Foundations of Mathematics, volume 103 (second printing), North-Holland, Amsterdam, 1985.

[3] Bollobás B. and S. Rasmussen, First cycles in random directed graph processes, Discrete Mathematics **75**, p 55-68, 1989.

[4] Church A, The calculi of lambda-conversion, Annals of Math. Studies, 6, Prinston Univ. Press, 1941.

[5] Eigen M., Self-organization of matter and evolution of biological macromolecules, Naturwissenschaften **58**, 465, 1971.

[6] Eigen M. and P. Schuster, The hypercycle - A principle of natural self-organization, Springer-Verlag, Heidelberg, 1979.

[7] Farmer D., S. Kauffman, and N. Packard, Autocatalytic replication of polymers, Physica D **22**, 50, 1986.

[8] Fontana W., Algorithmic Chemistry, in Artificial Life II, SFI Studies in the Sciences of Complexity, Vol. X, eds. Langton, C. et al., Addison-Wesley, Redwood City, 1991, p 159-209.

[9] Fredkin E., Digital mechanics, Physica D **45**, p 254-270, 1990.

[10] Goldberg D., Genetic algorithms in search, optimization, and machine learning, Addison-Wesley, Readings, Mass., 1989.

[11] Holland J., Adaptation in natural and artificial systems, University of Michigan Press, Ann Arbor, 1975.

[12] Hopcroft J. and J. Ullman, Introduction to automata theory, languages, and computation, Addison-Wesley (1979).

[13] Kauffman S., Autocatalytic sets of proteins, J. Theor. Bio. **119**, p 1-24, 1986.

[14] Knudsen C., R. Feldberg, and S. Rasmussen, Information Dynamics of Self-Programmable Matte, in: Complex Dynamics and Biological Evolution, eds. Mosekilde, E. and L. Mosekilde, Plenum Press, Series B: Physics Vol. 270, New York, 1991, p 223-245.

[15] Kleene S., Turing analysis of computability and major applications of it, in: The universal Turing machine: A half-century survey, R. Herken, ed., p 17-54, Oxford Univ. Press, 1988.

[16] Langton C., ed., Artificial life, Santa Fe Institute, Studies in the Sciences of Complexity, vol. IV, Addison Wesley, Redwood City, 1989.

[17] Langton C., C. Taylor, D. Farmer, and S. Rasmussen, eds., Artificial life II, Santa Fe Institute, Studies in the Sciences of Complexity, vol. X, Addison Wesley, Redwood City, 1991.

[18] Lindgren K. and M. Nordahl, Universal computation in simple one dimensional cellular automata, Complex Systems 4(3), p 299-318, (1990).

[19] McCaskill J., Preprint 1989, and private communication.

[20] Margolus N., Physics-like models of computation, Physica D **10**, p 81-95, 1984.

[21] Minsky M., Computation - finite and infinite machines, Prentice-Hall, 1972.

[22] Moore C., Unpredictability and undecidability in dynamical systems, Phys. Rev. Let., 64, p 2354-2357, 1990.

[23] Omohundro S., Modeling cellular automata with partial differential equations, Physica D **10**, 128-134, 1984.

[24] Post E., Formal reduction of the general reduction of the general combinatorial decision problem, Am. Jour. of Math., **65**, p 197-268, 1943.

[25] Rasmussen S., Toward a Quantitative Theory of the Origin of Life, in: Artificial Life, SFI Studies in the Sciences of Complexity, vol VI, ed. Langton C., Addison-Wesley, Redwood City, 1989, p 79-104.

[26] Rasmussen S., C. Knudsen, R. Feldberg, and M. Hindsholm, The Coreworld: Emergence and Evolution of Cooperative Structures in a Computational Chemistry, Physica **D** 42 (1990), p 111-134,

[27] Rasmussen S., C. Knudsen, and R. Feldberg, Dynamics of Programmable Matter, in Artificial Life II, SFI Studies in the Sciences of Complexity, Vol. X, eds. Langton, C. et al., Addison-Wesley, Redwood City, 1991, p 211-254.

[28] Rasmussen S., Aspects of Information, Life, Reality, and Physics, in Artificial Life II, SFI Studies in the Sciences of Complexity, Vol. X, eds. Langton, C. et al., Addison-Wesley, Redwood City, 1991, p 767-773.

[29] Ray T., An approach to the synthesis of life, in Artificial Life II, SFI Studies in the Sciences of Complexity, Vol. X, eds. Langton, C. et al., Addison-Wesley, Redwood City, 1991, p 371-408.

[30] Smith III A. R., Simple computation-universal cellular spaces, *JACM*, 18(3), 337-353, 1971.

[31] Sober E., Laerning from functionalism - prospects for strong artificial life, in Artificial Life II, SFI Studies in the Sciences of Complexity, Vol. X, eds. Langton, C. et al., Addison-Wesley, Redwood City, 1991, p 749-765.

[32] Spafford E. H., Computer Virus - A form of Artificial Life? in Artificial Life II, SFI Studies in the Sciences of Complexity, Vol. X, eds. Langton, C. et al., Addison-Wesley, Redwood City, 1991, p 727-745.

[33] Turing A., On computable numbers with applications to the Entscheidungs Problem, Proc. Lond. Math. Soc., **42**, p 230-265, 1936-1937.

[34] von Neumann J., Theory of self-reproducing automata, edited and completed by A. Burks, University of Illinois Press, 1966.