LA-UR 90-3924

CONF-9006299--2

LA-UR--90-3924

DE91 004851

TITLE: Functional Self-organization in Complex Systems

AUTHOR(S): Walter Fontana, Theoretical Division/Center for Nonlinear Studies
Los Alamos National Laboratory, Los Alamos, New Mexico

SUBMITTED TO: "Lectures in the Sciences of Complexity"
D.L. Stein, editor, Lectures Volume III
Santa Fe Institute Summer School 1990

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

MASTER

Functional Self-organization in Complex Systems

Walter Fontana

Theoretical Division and Center for Nonlinear Studies
Los Alamos National Laboratory
Los Alamos, NM 87545 USA

and

Santa Fe Institute
1120 Canyon Road
Santa Fe, NM 87501 USA

ABSTRACT

A novel approach to functional self-organization is presented. It consists of a universe generated by a formal language that defines objects (=programs), their meaning (=functions), and their interactions (=composition). Results obtained so far are briefly discussed.

1 INTRODUCTION

Nonlinear dynamical systems give rise to many phenomena characterized by a highly complex organization of phase space, e.g. turbulence, chaos, and pattern formation. The structure of the interactions among the objects described by the variables in these systems is usually fixed at the outset. Changes in the phase portrait occur as coefficients vary, but neither the basic qualitative relationships among the variables, nor their number is subject to change.

The class of systems I will be concerned with in this contribution is, in some sense, complementary. It contains systems that are "inherently constructive". By "constructive" I mean that the elementary interaction among objects includes the possibility of building new objects. By "inherently constructive" I want to emphasize that the generation of new objects from available ones is an intrinsic, specific, non-random property of the objects under consideration. It is *not* primarily caused by noise.

A prime example of such a system is chemistry. Molecules undergo reactions that lead to the production of new molecules. The formation of the product object is *instructed* by the interacting reactants. This is to be distinguished from a situation in which new objects are generated by chance events, as is the case with copying errors during the replication process of a DNA string (mutations).

Examples for complex systems belonging to the constructive category are chemistry, biological systems like organisms or ecosystems, as well as economies. Clearly, nonlinearities and noise occur all over the place. In this note I will be primarily concerned with the implications following from the constructive properties. An example of a dynamical system built on top of these properties will be given, but the formulation of a general theory (if it exists) that combines nonlinear dynamical *and* constructive aspects of complex systems is a major problem for the future.

The "manipulation" of objects through other objects, as it occurs with molecules, might in principle be reduced to the behavior of the fundamental physical forces relevant for the particular objects. Quantum mechanics is a "classical" example with respect to chemistry. At the same time, however, the very phenomenon of manipulation introduces a new level of description: it generates the notion of "functionality". Objects can be put into *functional* relations with each other. Such relations express which object produces which object under which conditions.

The main assumption of the present work is: *The action of an object upon other objects can be viewed as the application of a computable function to arguments from the functions' domain of definition (which can be other functions). Functional relations can then be considered in complete independence from their particular physical realization.*

There is no doubt that this is a very strong assumption, but such an abstraction is useful if we want to focus on a classification of functional relations in analogy to a classification of attractors in dynamical systems theory. It is also useful in defining toy models that capture the feed-back loop between objects and the functions acting on them defined by these very objects. It is such a loop that identifies complex constructive systems.

"Function" is a concept that - in some mathematical sense - is irreducible. Mathematics provides since 1936, through the works of Church and Kleene (1932-34), Gödel and Herbrand (1934), and Turing (1936), a formalization of the intuitive notion of "effective procedure" in terms of a complete theory of particular functions on the natural numbers: the partial recursive functions.

The following is a very brief attempt to explore the possibility of establishing a useful descriptive level of at least some aspects of constructive complex systems by viewing their objects as being machines = computers = algorithms = functions. For a more detailed exposition see [8,9].

2 WHAT IS A FUNCTION?

A function, f, can be viewed (roughly) in two ways:

- A function as an *applicative rule* refers to the *process* - coded by a definition - of going from argument to value.
- A function as a *graph*, refers to a set of ordered pairs such that if $(x, y) \in f$ and if $(x, z) \in f$, then $y = z$. Such a function is essentially a look-up table.

The first view stresses the computational aspect, and is at the basis of Church's λ-calculus (see for example [2]). The theory λ is a formalization of the notion of computability in precisely the same sense as the Turing machine and the theory of general recursiveness. Though its equivalence with Turing machines, λ is a very different and much more abstract approach.

Stated informally, λ defines inductively expressions consisting of variables. A variable is an expression, and every *combination* of expressions, wrapped in parentheses, is again an expression. Furthermore, a variable can be *substituted* by *any* other expression that follows the expression to which the variable belongs. An elegant notational (syntactic) structure provides a means for tagging the variables in an expression such as to define their scope.

In an "everyday" informal notation this means that if $f[x]$ and $g[x]$ are expressions, then $(f[x]g[x])$ is also an expression, and $(f[x]g[x])$ is equivalent to $(f[x := g[x]])$, where the latter denotes the expression that arises if every occurrence of x in $f[x]$ is replaced by the expression $g[x]$.

Intuitively, what is captured here is just the notion of "evaluating" a function by "applying" it to the argument. That is: consider $f[x]$ as denoting a function, then the *value* of that function when applied to the argument expression a is obtained by literally substituting a for x in $f[x]$ *and* performing all further substitutions that might become possible as a consequence of that action. If all substitutions have been executed, then an expression denoting the value of a under f has been obtained. In this way functions that can be identified with the natural numbers (numerals) as well as all computable operations on them, for example addition and multiplication, can be defined.

Three features of functions in λ are important for the following:

- Functions are defined recursively in terms of other functions: imagine functions as being represented by trees with variables at the leaf levels. This makes explicit that functions are "modular" objects, whose building blocks are again functions. This combinatorial representation, in which functions can be freely recombined to yield new functions, is crucial.
- Objects in λ can serve both as arguments or as functions to be applied to these arguments.
- There is no reference to any "machine" architecture.

Although the whole story is much more subtle, the preceeding paragraphs should convey the idea of "function" that will be used in the framework described in the following sections. The point is that under suitable mathematical conditions λ-objects and composition give rise to a reflexive algebraic structure.

Turing's completely different, but equivalent, approach to computability worked with the machine concept. It was this "hardware" approach that succeeded in convincing people that a formalization of "effective procedure" had been achieved. The existence of a universal Turing machine implies the logical interchangeability of hard- and software. Church's world of λ is entirely "software" oriented. Indeed, in spite of its paradigmatic simplicity it contains already many features of high-level programming languages. For a detailed account see [17].

3 THE MODEL

To set up a model that also provides a workbench for experimentation a representation of functions along the lines of the λ-calculus is needed. I have implemented a representation that is a somewhat modified and extremely stripped-down version of a toy-model of pure LISP as defined by Gregory Chaitin [3]. In pure LISP a couple of functions are pre-defined (6 in the present case). They represent primitive operations on trees (expressions); for example, joining trees or deleting subtrees. This speeds up and simplifies matters as compared to the λ-calculus in which one starts from "absolute zero" using only application and substitution. Moreover, I consider for the sake of simplicity only functions in one variable. The way functions act on each other thereby producing new ones is essentially identical to the formalism sketched for λ in the previous section.

I completely dispense with a detailed presentation of the language (which I refer to as "AlChemy": a contraction of Algorithmic Chemistry). Figure 1 and its caption give a simple example for an evaluation that should depict what it is all about. The interested reader is referred to [9].

The model is then built as follows.

1. Universe
   A universe is defined through the λ-like language. The language specifies rules for building syntactically legal ("well-formed") objects and rules for interpreting these structures as functions. In this sense the language represents the "physics". Let the set of all objects be denoted by $\mathcal{F}$.

2. Interaction
   Interaction among two objects, $f(x)$ and $g(x)$ is naturally induced by the language through function composition, $f(g(x))$. The evaluation of $f(g(x))$ results in a (possibly) new object $h(x)$. Interaction is clearly asymmetric. This can easily be repaired by symmetrizing. However, many objects like biological species or cell types (neurons, for example) interact in an asymmetric fashion. I chose to keep asymmetry.

   Note that "interaction" is just the name of a binary function $\phi(s,t)$ that sends any ordered pair of objects f and g into an object $h = \phi(f,g)$ representing the value of $f(g)$ (see figure 2 for an example). More generally, $\phi(s,t) : \mathcal{F} \times \mathcal{F} \mapsto \mathcal{F}$ could be *any* computable function, not necessarily composition, although composition is the most natural choice. The point is that whatever the "interaction"-function is chosen to be, it is itself evaluated according to the semantics of the language. Stated in terms of chemistry: it is the same chemistry that determines the properties of individual molecules and at the same time determines how two molecules interact.

3. Collision rule
   While "interaction" is intrinsic to the universe as defined above, the collision rule is not. The collision rule specifies essentially three arbitrary aspects:

   (a) what happens with f and g once they have interacted. These objects could be "used up", or they could be kept (information is not destroyed by its usage).

   (b) what happens with the interaction product h. Some interactions produce objects that are bound to be inactive no matter with whom they collide. The so called NIL-function is such an object: it consists of an empty expression. Several other constructs have the same effect, like function expressions that happen to lack any occurrence of the variable. In general such products are ignored, and the collision among f and g is then termed "elastic", otherwise it is termed "reactive".

   (c) computational limits. Function evaluation need not halt. The computation of a value could lead to infinite recursions. To avoid this, recursion limits, as well as memory and real time limitations have to be imposed. A collision has to terminate within some pre-specified limits, otherwise the "value" consists in whatever has been computed until the limits have been hit.

The collision rule is very useful for introducing boundary conditions. For example, every collision resulting in the copy of one of the collision partners might be ignored. The definition of the language is not changed at all, but identity functions would have now been prevented from appearing in the universe.

In the following I will imply that the interaction among two objects has been "filtered" by the collision rule. That is, the collision of f and g is represented by $\Phi(f,g)$ that returns $h = \phi(f,g)$ iff the collision rule accepts h (see item (b) above), otherwise the pair (f,g) is not in the domain of Φ.

4. System
To investigate what happens once an ensemble of interacting function "particles" is generated, a "system" has to be defined. The remaining sections will briefly consider two systems:

   - An iterated map acting on sets of functions.
   Let $\mathcal{P}$ be the power set, $2^{\mathcal{F}}$, of the set of all functions $\mathcal{F}$. Note that $\mathcal{F}$ is countable infinite, but $\mathcal{P}$ is uncountable. Let $\mathcal{A}_i$ denote subsets of $\mathcal{F}$, and let $\Phi[\mathcal{A}]$ denote the set of functions obtained by all $|\mathcal{A}|^2$ pair-interactions (i.e. pair-collisions) $\Phi(i,k)$ in $\mathcal{A}$, $\Phi[\mathcal{A}] = \{j : j = \Phi(i,k), (i,k) \in \mathcal{A} \times \mathcal{A}\}$. The map M is defined as

$$M : \mathcal{P} \mapsto \mathcal{P},\ \mathcal{A}_{i+1} = \Phi[\mathcal{A}_i]. \tag{1}$$

   Function composition induces a dynamics in the space of functions. This dynamics is captured by the above map M. An equivalent representation in terms of an interaction graph will be given in the next section.

   - A Turing gas.
   The Turing gas is a stochastic process that induces an additional dynamics over the nodes of an interaction graph. Stated informally, individual objects now acquire "concentrations" much like molecules in a test-tube mixture. However, the graph on which this process lives changes as reactive collisions occur. Section 5 will give a brief survey on experiments with the Turing gas.

4 AN ITERATED MAP AND INTERACTION GRAPHS

The interactions between functions in a set $\mathcal{A}$ can be represented as a directed graph G. A graph G is defined by a set $V(G)$ of vertices, a set $E(G)$ of edges, and a relation of incidence, which associates with each edge two vertices (i,j). A directed graph, or digraph, has a direction associated with each edge. A labelled graph has in addition a label k assigned to each edge (i,j). The labelled edge is denoted by (i,j,k).

The action of function $k \in \mathcal{A}$ on function $i \in \mathcal{A}$ resulting in function $j \in \mathcal{A}$ is represented by a directed labelled edge (i,j,k):

$$(i,j,k):\ i \xrightarrow{k} j \quad i,j,k \in \mathcal{A}. \tag{2}$$

Note that the labels k are in $\mathcal{A}$. The relationships among functions in a set are then described by a graph G with vertex set $V(G) = \mathcal{A}$ and edge set $E(G) = \{(i,j,k) : j = k(i)\}$.

A useful alternative representation of an interaction is in terms of a "double-edge",

$$(i,j,k): \; i \xrightarrow{(i,k)} j \xleftarrow{(i,k)} k \quad i,j,k \in \mathcal{A}, \tag{3}$$

where the function k acting on i and producing j has now been connected to j by an additional directed edge. The edges are still labelled, but no longer with an element of the vertex set. The labels (i,k) are required to uniquely reconstruct the edge set from a drawing of the graph. The graph corresponding to a given edge set is obviously uniquely specified. Suppose, however, that a function j is produced by two different interactions. The corresponding vertex j in the graph then has four inward edges. Uniquely reconstructing the edge set, or modifying the graph, for example by deleting a vertex, requires information about which pair of edges results from the same interaction. Some properties of the interaction graph can be obtained while ignoring the information provided by the edge labels. The representation in terms of double-edges (i,j,k) has the advantage to be meaningful for any interaction function Φ mapping a pair of functions (i,k) to j, and not only for the particular Φ representing chaining. The double-edge suggests that both, i as well as k, are needed to produce j. In addition, the asymmetry of the interaction is relegated to the label: (i,k) implies an interaction $\Phi(i,k)$ as opposed to $\Phi(k,i)$. This representation is naturally extendable to n-ary interactions $\Phi(i_1, i_2, \ldots, i_n)$. In the binary case considered here every node in G must therefore have zero or an even number of incoming edges.

The following gives a precise definition of an interaction graph G. As in equation (1) let $\Phi[\mathcal{A}]$ denote the set of functions obtained by all possible pair-collisions $\Phi(i,k)$ in $\mathcal{A}$, $\Phi[\mathcal{A}] = \{j : j = \Phi(i,k), (i,k) \in \mathcal{A} \times \mathcal{A}\}$. The interaction graph G of set $\mathcal{A}$ is defined by the vertex set

$$V(G) = \mathcal{A} \cup \Phi[\mathcal{A}] \tag{4}$$

and the edge set

$$E(G) = \{(i,j,k) : \; i,k \in \mathcal{A}\,,\; j = \Phi(i,k)\} \tag{5}$$

The graph G is a function of $\mathcal{A}$ and Φ, $G[\mathcal{A}, \Phi]$. The action of the map

$$M : \mathcal{A}_{i+1} = \Phi[\mathcal{A}_i] \tag{6}$$

on a vertex set $\mathcal{A}_i$ leads to a graph representation of M. Let

$$G^{(i)}[\mathcal{A}, \Phi] := G[\Phi^i[\mathcal{A}], \Phi] \tag{7}$$

denote the i-th iteration of the graph G starting with vertex set $\mathcal{A}$; $G^{(0)} = G$.

A graph G and its vertex set $V(G)$ are closed with respect to interaction, when

$$\Phi[V(G)] \subseteq V(G), \tag{8}$$

otherwise G and $V(G)$ are termed innovative.

Consider again the map M, equation (6). What are the fixed points of $\Phi[\cdot]$? $\mathcal{A} = \Phi[\mathcal{A}]$ is equivalent to (1) $\mathcal{A}$ is closed with respect to interaction, and (2) the set $\mathcal{A}$ reproduces itself under interaction. That is,

$$\forall j \in \mathcal{A}, \; \exists \; i,k \in \mathcal{A} \text{ such that } j = \Phi(i,k). \tag{9}$$

Condition (9) states that all vertices of the interaction graph G have at least one inward edge (in fact, two or any even number). Such a self-maintaining set will also be termed "autocatalytic", following M.Eigen [5] and S.A.Kauffman [11] who recognized the relevance of such sets with respect to the self-organization of biological macromolecules.

Consider a set $\mathcal{F}_i$ for which (9) is still valid, but which is not closed with respect to interaction. $\mathcal{F}_{i+1}$ obviously contains $\mathcal{F}_i$, because of (9), and in addition it contains the set of new interaction products $\Phi[\mathcal{F}_i] \setminus \mathcal{F}_i$. These are obviously generated by interactions within $\mathcal{F}_i \in \Phi[\mathcal{F}_i]$. Therefore (9) also holds for the set $\Phi[\mathcal{F}_i]$, implying that the set $\mathcal{F}_{i+1}$ is autocatalytic. Therefore, if $\mathcal{A}$ is autocatalytic, it follows that

$$G[\mathcal{A}, \Phi] \supseteq G^{(1)}[\mathcal{A}, \Phi] \supseteq G^{(2)}[\mathcal{A}, \Phi] \supseteq \ldots \supseteq G^{(i)}[\mathcal{A}, \Phi] \supseteq \ldots . \tag{10}$$

In the case of strict inclusion let such a set be termed "autocatalytically self-extending". Such a set is a special case of innovation, in which

$$\Phi[V(G)] \supseteq V(G) \tag{11}$$

holds, with equality applying only at closure of the set.

An interesting concept arises in the context of finite, closed graphs. Consider, for example, the autocatalytic graph G in figure 3b, and assume that G is closed. The autocatalytic subset of vertices $V_1 = \{A, B, D\}$ induces an interaction graph $G_1[V_1, \Phi]$. Clearly, $G[V, \Phi] = G_1^{(2)}[V_1, \Phi]$, which means that the autocatalytic set V_1 regenerates the set V in two iterations. This is not the case for the autocatalytic graph shown in figure 3a. More precisely, let G be a finite interaction graph, and let $G_\alpha \subseteq G$ be termed a "seeding set" of G, if

$$\exists\, i, \text{ such that } G \subseteq G_\alpha^{(i)}, \tag{12}$$

where equality must hold if G is closed. Seeding sets turn out to be interesting for several reasons. For instance, in the next section a stochastic dynamics (Turing gas) will be induced over an interaction graph. If a system is described by a graph that contains a small seeding set, the system becomes less vulnerable to the accidental removal of functions. In particular cases a seeding set can even turn the set it seeds into a limit set of the process. Such a case arises when every individual function f_i in $\mathcal{A}$ is a seeding set of $\mathcal{A}$:

$$\begin{aligned} f_{i+1} &= \Phi(f_i, f_i) \quad i = 1, 2, \ldots, n-1 \\ f_1 &= \Phi(f_n, f_n). \end{aligned} \tag{13}$$

Furthermore, suppose that G is finite, closed and autocatalytic. It follows from the above that all seeding sets G_α must be autocatalytically self-extending, as for example in figure 3b. If G is finite, closed, but not autocatalytic, there can be no seeding set. Being closed and not autocatalytic implies $V(G^{(2)}) \subset V(G)$. The vertices of G that have no inward edges are lost irreversibly at each iteration. Therefore, for some i either $G^{(i)} = \emptyset$, or $G^{(i)}$ becomes an autocatalytic subset of G.

In the case of innovative, not autocatalytic sets, i.e. sets for which

$$\Phi[\mathcal{A}] \not\subseteq \mathcal{A} \wedge \Phi[\mathcal{A}] \not\supseteq \mathcal{A} \tag{14}$$

holds, no precise statement can be made at present.

A digraph is called connected if for every pair of vertices i and j there exists at least one directed path from i to j and at least one from j to i. An interaction graph G that is connected not only implies an autocatalytic vertex set, but in addition depicts a situation in which there are no "parasitic" subsets. A parasitic subset is a collection of vertices that has only incoming edges, like the single vertices C and E in figure 3b, or the set $\{C, E\}$ in figure 3a. As the name suggests, a parasitic subset is not cooperative, in the sense that it does not contribute to generate any functions outside of itself.

All the properties discussed in this section are independent of the information provided by the edge labels (in the double-edge representation). Note furthermore, that the above discussion is independent of any particular model of "function". It never refers to the implementation in the LISP-like AlChemy. The representation of function in terms of this particular language is used in the simulations reported briefly in the next section.

5 A TURING GAS

The interaction graph, and, equivalently, the iterated map, describe a dynamical system induced by the language on the power set of functions. This graph dynamics is now supplemented by a mass action kinetics leading to a density distribution on the set of functions in a graph. The kinetics is induced through a stochastic process termed "Turing gas".

A Turing gas consists of a fixed number of function particles that are randomly chosen for pairwise collisions. In the present scheme a reactive collision keeps the interaction partners in addition to the reaction product. When a collision was reactive the total number of particles increases by one. To keep the number constant, one particle is chosen at random and erased from the system. This mimics a stochastic unspecific dilution flux. The whole system can be compared to a well stirred chemical flow-reactor.

Three versions of the Turing gas have been studied. In one version the time evolution of the gas is observed after its initialization with N (typically $N = 1000$) randomly generated functions. In the second version the collision rule is changed to forbid reactions resulting in a copy of one of the collision partners. In the third version the gas is allowed to settle into a quasi-stationary state, where it is perturbed by injecting new random functions.

The following summarizes very briefly some of the results.

(1) plain Turing gas

Ensembles of initially random functions self-organize into ensembles of specific functions sustaining cooperative interaction pathways. Self-replicators (functions that copy themselves, but not the others *present in the system*), parasites (see section 4), general copy functions (identity functions), as well as partial copiers (functions that copy some, but not all functions they interact with) shape the dynamics of the system. The "innovation rate", i.e. the frequency of collisions that result in functions not present in the system, decreases with time indicating a steady closure with respect to interactions. If the stochastic process is left to itself after injecting the initial functions, fluctuations will eventually drive it into an absorbing barrier characterized by either a single replicator type, or by a possibly hetergeneous mixture of non-reactive functions ("dead system"), or by a self-maintaining set where each individual function species is a seeding set (section 4). The system typically exhibits etremely long transients characterized by mutually stabilizing interaction patterns. Figure 4 shows an interaction graph (in a slightly different representation than described in section 4; see caption) of a very stable self-maintaining set that evolved during the first $3 \cdot 10^5$ collisions starting from 1000 random functions. All functions present in the system differ from that initial set. The numbers refer to the function expressions (not shown) as they rank in lexicographic order. Function 17 is an identity function, although – for the sake of a less congested picture – only the self-copying interaction is displayed. Patterns like those in figure 4 often include a multitude of interacting self-maintaining sets. In figure 4, for example, deleting group I on the upper left still leaves a self-maintaining system (group II). Several other parts can be deleted while not destroying the cooperative structure. Sometimes these subsets are disconnected from each other with respect to interconversion pathways (solid arrows), but connected with respect to functional couplings (dotted lines). Figure 4 shows two groups of functions (indicated by I and II) that are not connected by transformation pathways (solid arrows). That is: no function of I is acted upon by any other function in the system such that it is converted into a function of group II – and *vice versa*. Group I, however, depends on group II for survival. The introduction of a "boundary" between I and II, cutting off all functional couplings among them, would destroy group I, but not group II.

(2) Turing gas without copy reactions

Copy reactions, i.e. interactions of the type $f(g) = g$ or f, strongly influence the patterns that evolve in the Turing gas as described above. Forbidding copy reactions (by changing the collision rule, section 3) results into a rather different type of cooperative organization as compared to the case in which copy reactions and therefore self-replicators were allowed. The system switches to functions based on a "polymeric" architecture that entertain a closed web of mutual synthesis and degradation reactions. The individual functions are usually organized into disjoint subsets of polymer families based on distinct monomers. As in the case of copy reactions these subsets interact along specific functional pathways leading to a cooperativity at the set level. Figure 5a and 5b shows an example of two interacting polymer families. Neither family could survive without the other. Due to the polymeric structure of the functions the Turing gas remains highly innovative. A much higher degree of diversity and stability is achieved than in systems that are dominated by individual self-replicators. The high stability of the system is due to very small seeding sets. For example, everything in the system shown in figure 5a and 5b follows from the presence of monomer 1 of type A (fig. 5a) and monomer 1 of type B (fig. 5b). Almost the whole system can be erased, but as long as there is one monomer A1 and one monomer B1 left, the system will be regenerated.

(3) Turing gas with perturbations

The experiments described so far kept the system "closed" in the sense that at any time instant the system's population can be described by series of compositions expressed in terms of the initially present functions. An open system is modeled by introducing new random functions that perturb a well established ecology. In the case without copy reactions the system underwent transitions among several new quasi-stationary states (metastable transients) each characterized by an access to higher diversity. Systems with copy reactions were more vulnerable to perturbations and lost in the long run much of their structure.

A detailed analysis is found in [9].

6 CONCLUSIONS

The main conclusions are:

1. A formal computational language captures basic qualitative features of complex adaptive systems. It does this because of

   (a) a powerful, abstract and consistent description of a system at the "functional" level, due to an unambiguous mathematical notion of function.

(b) a finite description of an infinite (countable) set of functions, therefore providing a potential for functional open-endedness.

(c) a natural way of enabling the construction of new functions through a consistent definition of interaction between functions.

2. Populations of individuals that are both, an object at the syntactic level and a function at the semantic level, give rise to the spontaneous emergence of complex, stable, and adaptive interactions among their members.

7 QUESTIONS AND FUTURE WORK

The main questions and directions for the future can be summarized as follows.

1. Is there an equivalent of a "dynamical systems theory" for functional interactions? Can the dynamical behavior of the iterated map, eq.(1), be characterized? Can examples be found that exhibit attractors other than fixed points? Can a classification of all finite self-maintaining sets of unary functions be made?

2. What is beyond replicator (Lotka-Volterra) equations? The standard replicator equation [10,6] on the simplex $S_n = \{x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n : \sum x_i = 1, x_i > 0\}$,

$$\dot{x}_i = x_i\Big(\sum_j a_{ij}x_j - \sum_{k,l} a_{kl}x_k x_l\Big) \quad i = 1, \ldots, n, \tag{15}$$

considers objects i that are individual self-replicators. The Turing gas deals with finite populations, and represents a stochastic version of its deterministic, infinite population counterpart [14]

$$\dot{x}_i = \sum_{j,k} a_{ijk}x_j x_k - \sum_{k,l,m} a_{klm}x_k x_l x_m \quad i = 1, \ldots, n, \tag{16}$$

where in the present case the entry $a_{ijk} = 1$ iff function k acting on function j produces function i, and $a_{ijk} = 0$ otherwise, and x_i, $0 \leq x_i \leq 1$, represents the frequency of function i in the system.
What can be said about the behavior of equation (16)? What can be deduced from it for the finite population Turing gas? Note that the use of a formal language, like λ, allows a finite description of a particular instance of an *infinite* matrix $a_{ijk}, i, j, k = 1, 2, \ldots$.

3. An obvious extension of the present work is the multivariable case. The restriction to unary functions implied that only binary collisions could be considered. n-ary functions lead to $(n+1)$-body interactions. Suppose the interaction is still given by function composition. A two variable function $f(x, y)$ then interacts with functions g and h (in this order) by producing $i = f(g, h)$. f acts with respect to any pair g and h precisely like a binary interaction law expression Φ. However, f can now be modified through interactions with other components of the same system. This might have significant consequences for the architecture of organizational patterns that are likely to evolve. The extension to n variables is currently in preparation.

4. Future work includes a systematic investigation of the system's response to noise (section 5, item 3), either in form of a supply of random functions, or in form of a "noisy

evaluation" of functions. What are the adaptive properties of the Turing gas?

5. If some of the questions above are settled, then an extension to more sophisticated, typed languages (distinction among various object/data "types") that enable a compact codification of more complicated – even numerical – processes could be envisioned; always keeping in mind that processes should be able to construct new processes by way of interaction. The combination with a spatial extension or location of these processes then leads to Chris Langton's [12] vision of a "process gas". I leave it to the reader to further speculate where such intriguing tools might lead.

8 RELATED WORK

The coupling of a dynamics governing the topology of an interaction graph with a dynamics governing a frequency distribution over its vertices is a common situation in biological systems. The immune system [4], development (ontogenesis) [15], and prebiotic molecular evolution [1] are but a few areas in which some modeling has been done.

The approach sketched here is related in particular to the pioneering work of D.Farmer *et al.* [7], R.Bagley *et al.* [1], S.Rasmussen *et al.* [16] and J.McCaskill [13].

Farmer and Bagley consider a system of polymers, intended to be polynucleotides or polypeptides, each of which specifically instructs (and catalyzes) the condensation of two polymers into one or the splitting of one into two. This sets up the constructive part of the system and the related dynamics of interaction graphs. Farmer and Bagley then describe the production rate of individual polymers in terms of enzyme kinetics by differential equations, providing a (nonlinear) dynamical system living on the interaction graphs. This approach represents one of the most advanced attempts to model a specific stage in prebiotic evolution.

Rasmussen and McCaskill made a first step towards abstraction, and the approach described here was prompted by their investigations. Rasmussen's system consists of generalized assembler code instructions that interact in parallel inside a controlled computer memory giving rise to cooperative phenomena. This intriguing system lacks, however, a clear cut and stable notion of functionality, except at the individual instruction level. McCaskill uses binary strings to encode transition-table machines of the Turing type that read and modify bit strings.

A model like the present Turing gas cannot provide much *detailed* information about a particular *real* complex system whose dynamics will highly depend on the physical realization of the objects as well as on the scheme by which the functions or interactions are encoded into these objects. Nevertheless, many phenomena that emerge, for example, in Farmer's and Bagley's polymer soup appear again within the approach described here. The hope then is that an abstraction cast purely in terms of functions might enable a quite general mathematical classification of cooperative organization. The question is if such an abstract level of description still can capture *principles* of complex *physical* systems. How much – in the case of complex systems – can we abstract from the "hardware" until a theory looses any explanatory power? I think the fair answer at present is: we don't know.

ACKNOWLEDGEMENTS

This work is ultimately the result of many "reactive collisions" with John McCaskill, David Cai, Steen Rasmussen, Wojciech Zurek, Doyne Farmer, Norman Packard, Chris Langton, Jeff Davitz, Richard Bagley, Stephanie Forrest, David Lane, and Stuart Kauffman. Thanks to all of them!

REFERENCES

1. R.J. Bagley, J.D. Farmer, S.A. Kauffman, N.H. Packard, A.S. Perelson, and Stadnyk. I.M. *BioSystems*, 23:113–138, 1989.

2. H.P. Barendregt. *The Lambda Calculus.* Studies in Logic and the Foundations of Mathematics, volume 103. North-Holland, Amsterdam, 1984.

3. G.J. Chaitin. *Algorithmic Information Theory.* Cambridge University Press, Cambridge, 1987.

4. R. deBoer and A. Perelson. *J. Theor. Biol.*, in press, 1990.

5. M. Eigen. *Naturwissenschaften*, 10, 1971.

6. M. Eigen and P. Schuster. *The Hypercycle.* Springer, Berlin, 1979.

7. J.D. Farmer, S.A. Kauffman, and N.H. Packard. *Physica D*, 22:50–67, 1986.

8. W. Fontana. *Algorithmic Chemistry.* Technical Report LA-UR 90-1959, Los Alamos National Laboratory, 1990. to appear in *Artificial Life II*, Proceedings of the second Artificial Life Workshop, editor C.G.Langton, Addison-Wesley.

9. W. Fontana. *Turing Gas: A New Approach to Functional Self-organization.* Technical Report LA-UR 90-3431, Los Alamos National Laboratory, 1990. submitted to Physica D.

10. J. Hofbauer and K. Sigmund. *The Theory of Evolution and Dynamical Systems.* Cambridge University Press, LMSST 7, Cambridge, 1988.

11. S.A. Kauffman. *J. Theor. Biol.*, 119:1–24, 1986.

12. C.G. Langton. 1990. personal communication.

13. J.S. McCaskill. 1990. in preparation.

14. J.H. Miller, W. Fontana, and P. Schuster. Towards a mathematics of a Turing gas. 1990. in preparation.

15. E. Mjolsness, D.H. Sharp, and J. Reinitz. *A Connectionist Model of Development.* Technical Report YALEU/DCS/RR-796, Yale University, 1990.

16. S. Rasmussen, C. Knudsen, R. Feldberg, and M. Hindsholm. *Physica D*, 42:111–134, 1990.

17. B.A. Trakhtenbrot. Comparing the Church and Turing approaches: two prophetical messages. In R.Herken, editor, *The Universal Turing Machine: A Half-Century Survey*, pp. 603–630. Oxford University Press, 1988.

FIGURE CAPTIONS

Fig. 1. Evaluation example.
The value of the expression ((+ a) (- a)) is computed when the variable a takes on the value ((* a a) (+ a)) . The interpretation process follows the tree structure until it reaches an atom (leaf). In this case it happens at depth 2. The atoms are evaluated: the operators "+" and "-" remain unchanged, while the value of a is given by ((* a a) (+ a)) . The interpreter backs up to compute the values of the nodes at the next higher level using the values of their children. The value of the left node at depth 1 is obtained by applying the unary "+"-operator to its sibling (which has been evaluated in the previous step). The "+" operation returns the first subtree of the argument, (* a a) in this case. Similarly, the value at the right depth 1 node is obtained by applying the unary "-"-operator to its argument ((* a a) (+ a)) . The "-" operation deletes the first subtree of its argument returning the remainder, (+ a) . The interpreter has now to assign a value to the top node. The left child's value is an expression representing again a function. This function, (* a a) , is labelled as f in the figure. Its argument is the right neighbor sibling, (+ a) , labelled as g. Evaluating the top node means applying f to g. This is done by evaluating f while assigning to its variable a the value g. The procedure then recurs along a similar path as above, shown in the box. The result of $f(g)$ is the expression ((+ a) (+ a)) . This is the value of the root (level 0) of the original expression tree, and therefore the value of the whole expression, given the initial assignment. The example can be interpreted as "function ((+ a) (- a)) applied to function ((* a a) (+ a)) ".

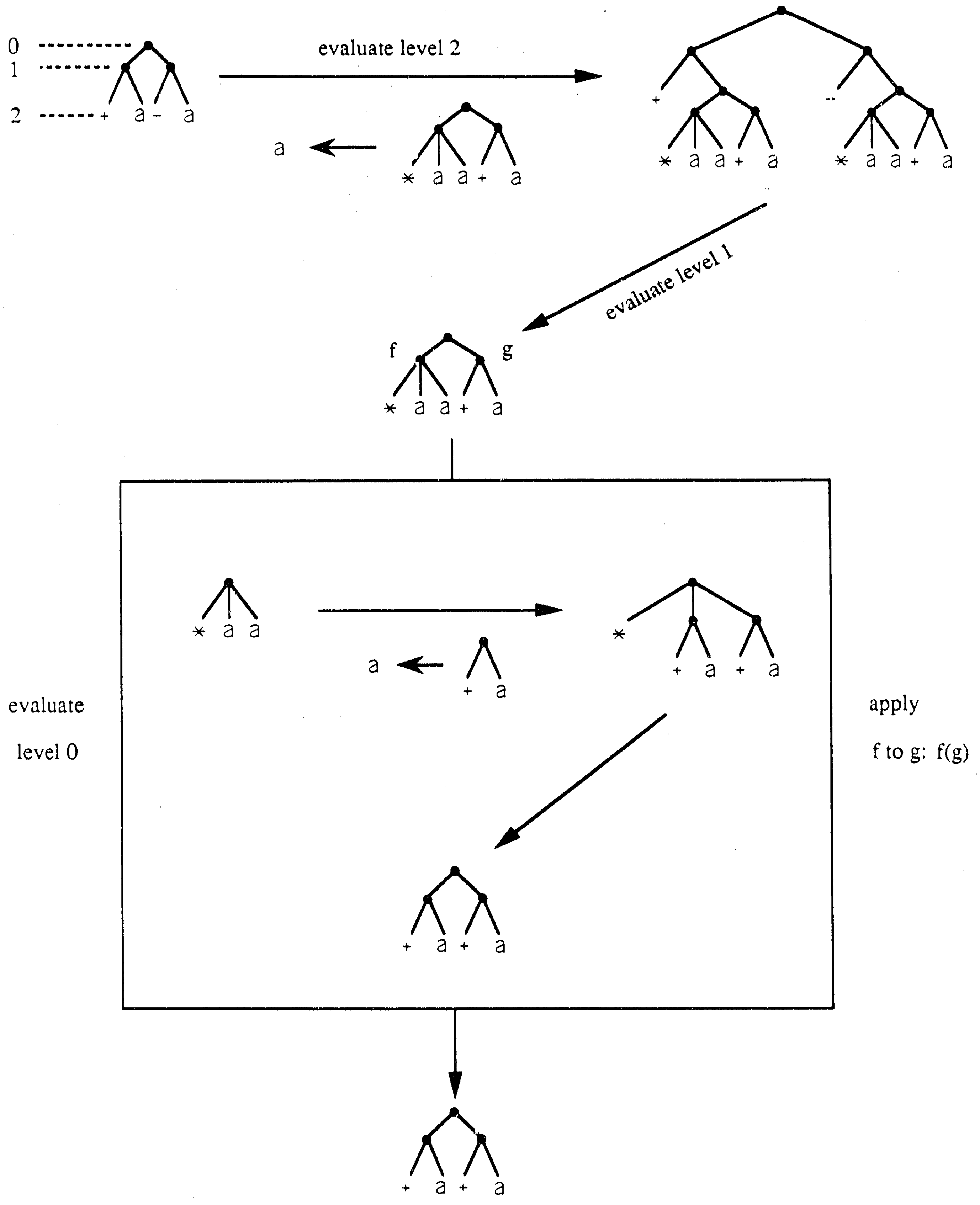
Fig. 2. Interaction between functions.
Two algorithmic strings (top) represented as trees interact by forming a new algorithmic string (middle) that corresponds to a function composition. The new root with its two branches and '-operators is the algorithmic notation for composing the functions. The action of the unary '-operator ("quote"-operator) consists in preventing the evaluation of its argument. The interaction expression is evaluated according to the semantics of the language and produces an expression (bottom) that represents a new function.

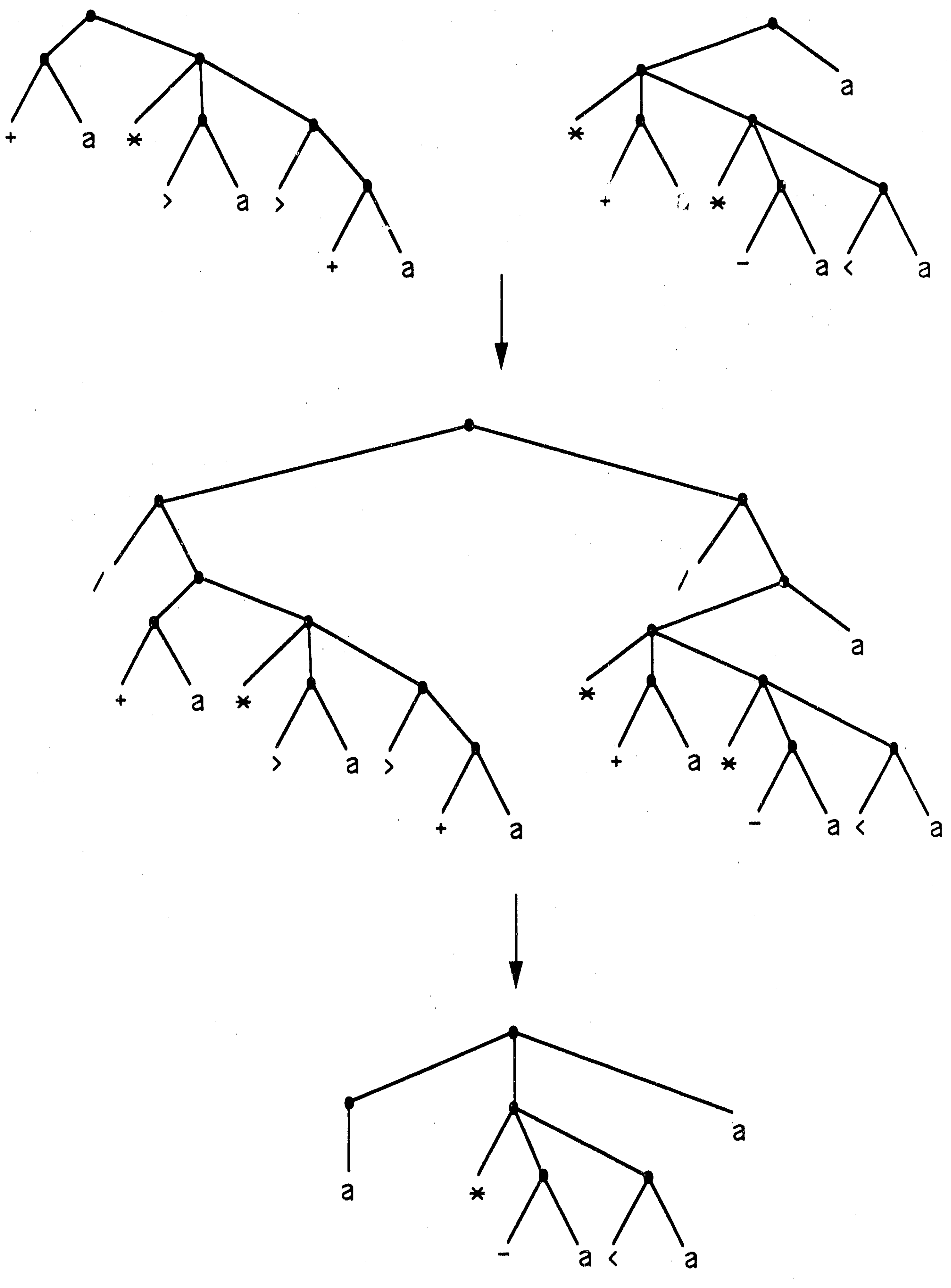
Fig. 3. Interaction graph and seeding set.
Two self-maintaining (autocatalytic) graphs. As outlined in section 4, a particular function, say D, is produced through interaction between A and B. Therefore D has two incoming edges from A and B. The edge labels are omitted. The dotted line indicates that B is applied to the argument A (solid line). The graphs are self-maintaining because every vertex has incoming edges. The lower graph, (b), can be regenerated from the vertex subset $\{A, B, D\}$, in contrast to the upper graph, (a). Both contain parasitic subsets: $\{E, C\}$ in (a), and $\{C\}$, $\{E\}$ in (b).

Fig. 4. Interaction graph of a metastable Turing gas transient.
The interaction graph of the functions present in the system after $3 \cdot 10^5$ collisions is shown. The system started with 1000 random functions, and conserves the total number of particles (1000). The numbers denote the individual functions according to their lexicographic ordering (not shown). Capital letters denote sets, where $A = \{1, 2, 3, 4\}$, $B = \{5, 6, 7, 8\}$, $C = \{9, 10\}$, $E = \{12, 13, 14, 15, 16\}$, and $E_1 = \{12, 13\} \in E$. Solid arrows indicate transformations, dotted lines functional couplings. A dotted line originates in a function, say k, and connects (filled circle) to a solid arrow, whose head is j and whose tail is i. This is to be interpreted as $j = k(i)$. Large filled circles

indicate membership in a particular set. Function 17 is an identity function. Note: all dotted lines and solid arrows that result from 17 copying everything else in addition to itself have been omitted. The function set is closed with respect to interaction.

Fig. 5. Interaction graph of a metastable Turing gas state without copy-reactions.
The figures show the interaction pathways among two polymer families established after $5 \cdot 10^5$ collisions starting from random initial conditions. The tree structure of the functions is displayed. The leaves are "monomers" representing the functional group indicated at the bottom of each graph. Solid arrows indicate transformations operated by function(s) belonging to the family denoted by the arrow label(s). Stars in the transformation pathways represent functions that were not present in the system at the time of the snapshot ("innovative reactions"). Due to the polymeric architecture of the functions, the system remains highly innovative. There are always (at least) two functions in the system that a polymerizing function (like the monomer 1 in 5a) can combine in order to produce a third one not in the system. The "space limit" tag in figure 5b indicates that the corresponding reaction product would hit the length limitation imposed on each individual function expression (300 characters in the present case).


0
1
2
evaluate level 2
evaluate level 1
f
g
evaluate
level 0
apply
f to g: f(g)

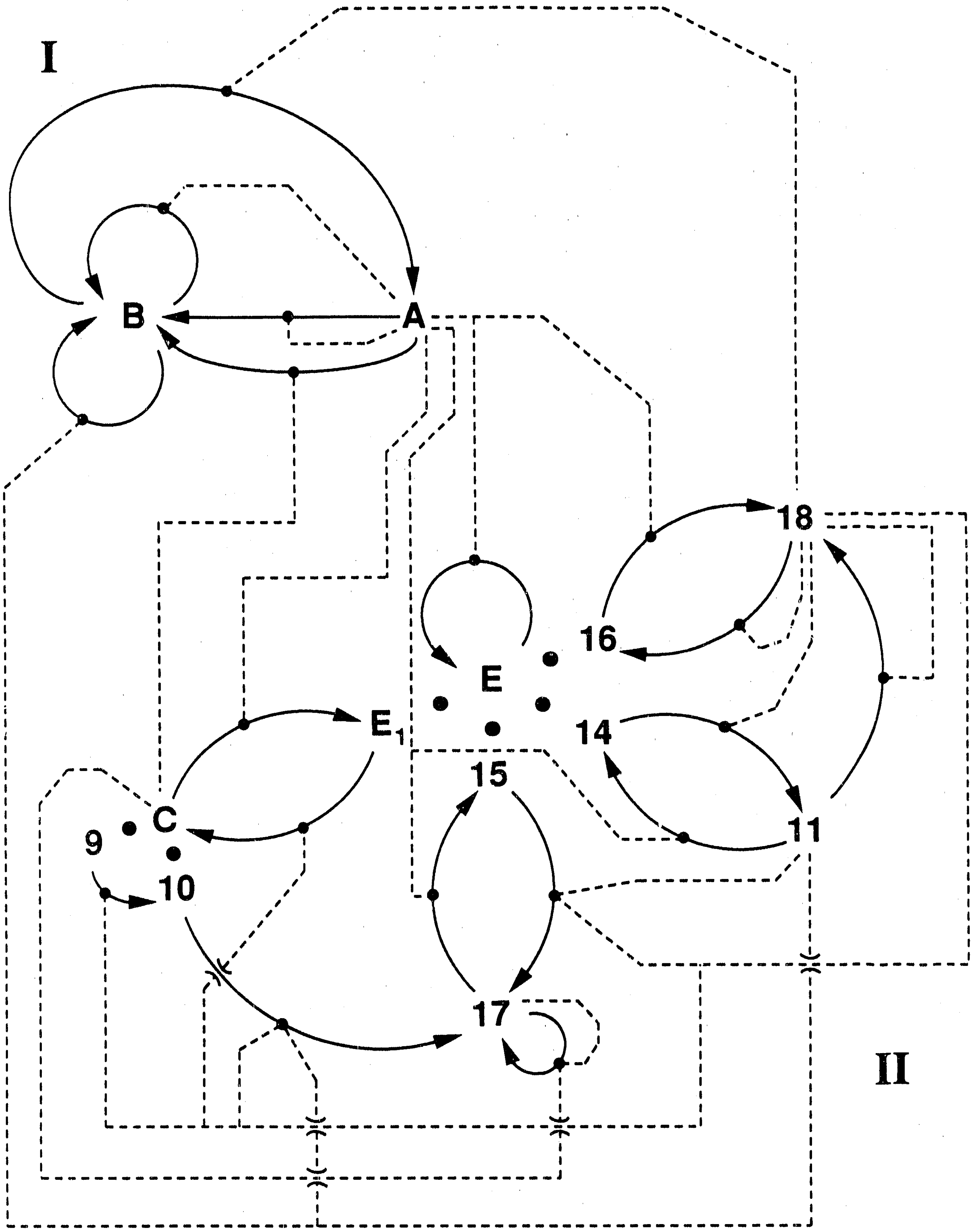

+
a
*
>
a
>
+
a
a
*
+
a
*
-
a
<
a
/
/
a
*
-
a
<
a

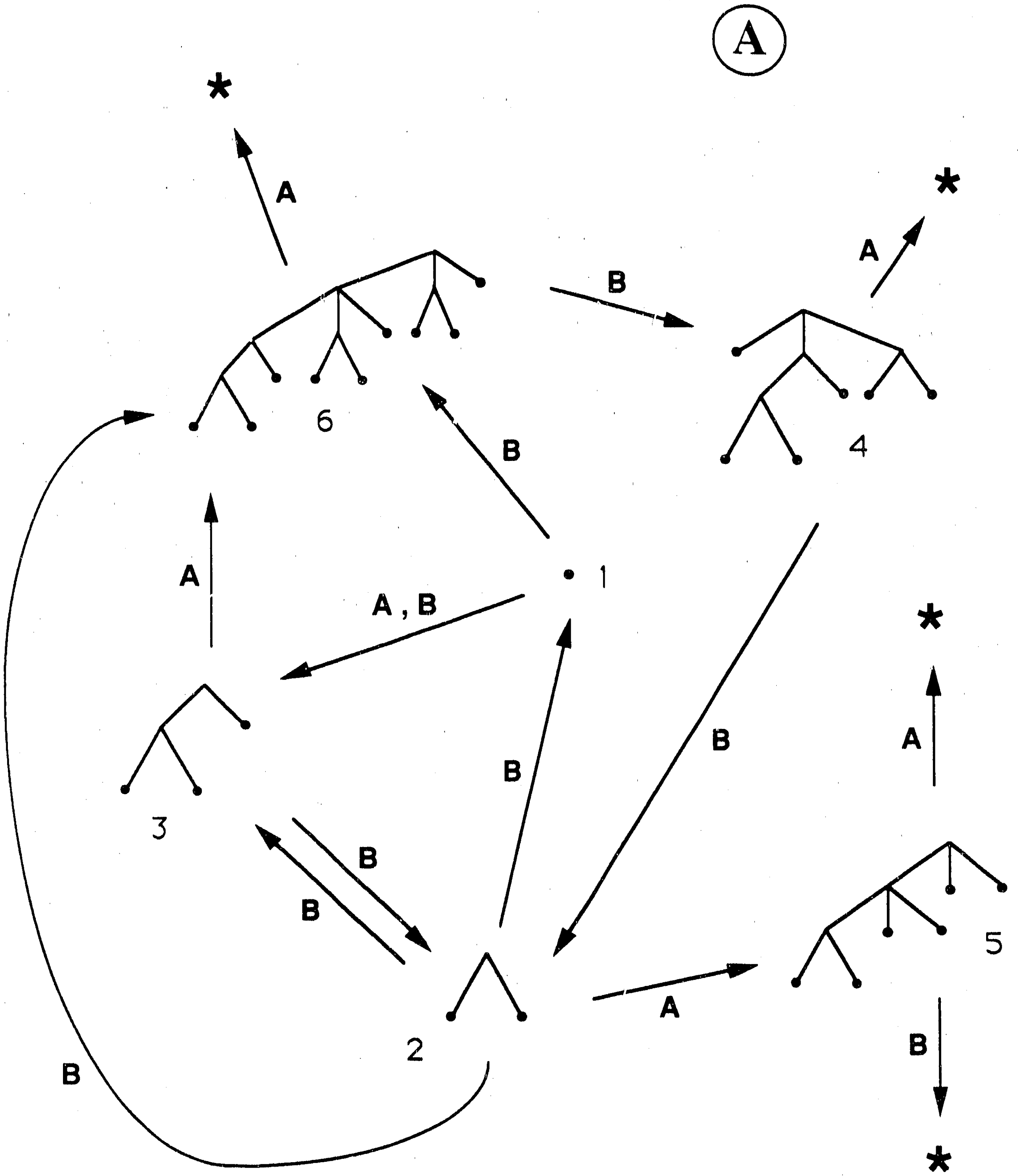
$$\begin{tikzcd}
& D \arrow[r] \arrow[dl, dashed, bend right] \arrow[dd] & E \arrow[dd, dashed, shift left] \\
A \arrow[ur] \arrow[dr, dashed, bend right] & & \\
& B \arrow[ul] \arrow[r] \arrow[uu, dashed, bend right] & C \arrow[uu, dashed, shift left]
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
& D \arrow[r] \arrow[dl, dashed, bend right] \arrow[dd] & E \arrow[dd, dashed] \\
A \arrow[ur] \arrow[dr, dashed, bend right] & & \\
& B \arrow[ul] \arrow[r] \arrow[uu, dashed, bend right] \arrow[uur, dashed] & C
\end{tikzcd}$$

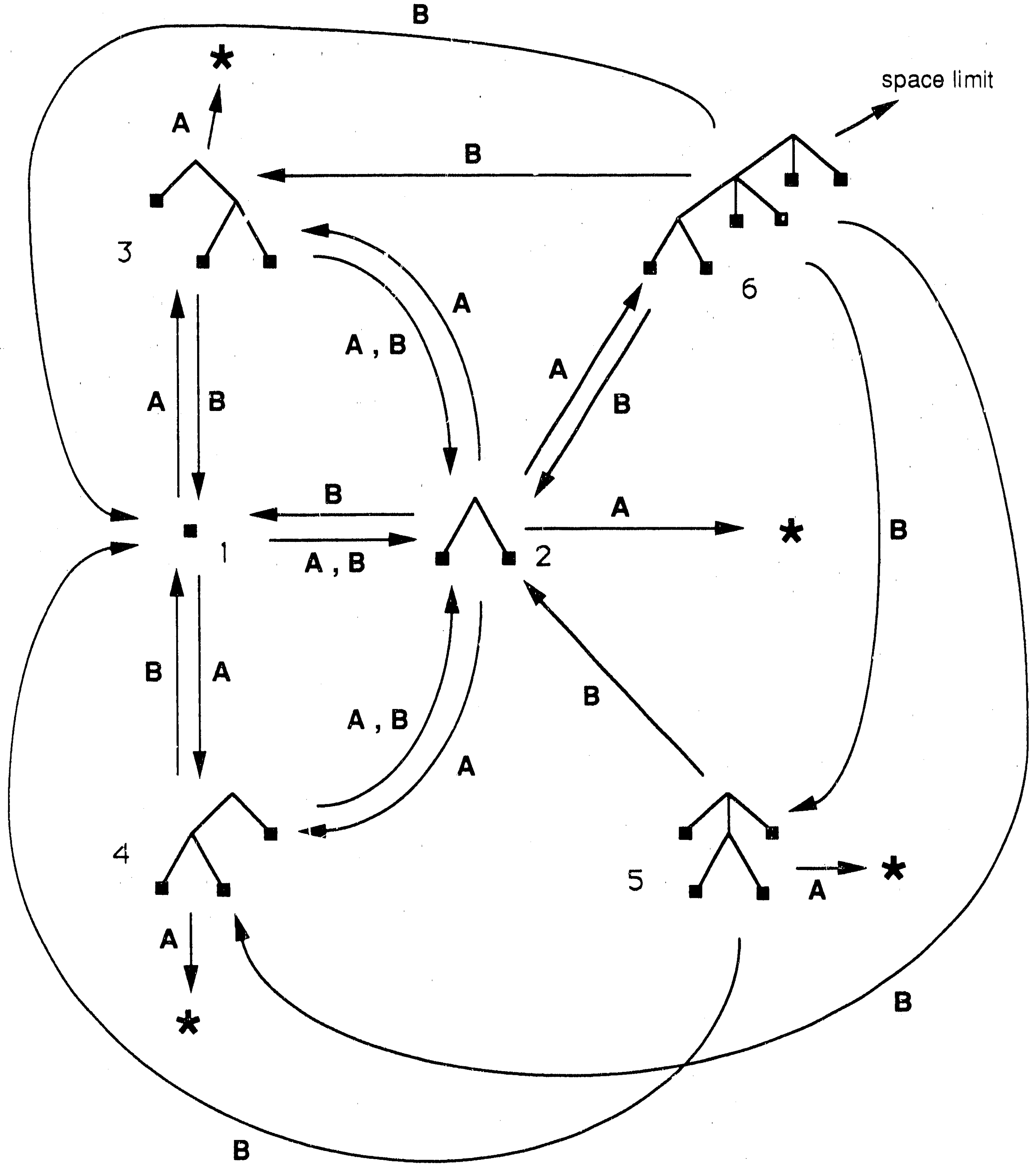
(b)


I
B
A
E
16
18
E1
14
15
C
9
10
11
17
II



● = (⋆(⋆aa)a)

B



■ = (<(*(>(*aa))(+a)))