FOURTH WINTER SCHOOL (1976)

ENVIRONMENTALLY CONDITIONED SELF-REPRODUCTION

by

Petr KŮRKA

In the paper we consider a model describing some features of relationship between biological organisms and their environment. We suppose the organism and its environment are made from the same stuff. The stuff consists of various elements with different properties. The elements may be grouped into assemblies, and these assemblies, when in contact,may act on each other, this action being determined by the elements involved and ways, they are grouped together. Given a sufficient quantity of elements, they may be put together in a great many ways, and it is probable that some of these assemblies have quite remarkable properties, e.g. self-reproduction.

In mathematical formalization, elements are characters of an alphabet $\mathcal{a}$ (containing all capital letters, digits, and some special symbols), assemblies are words of $\mathcal{a}$, and the relation "to be in contact" is realized by an antireflexive relation. So far we have a system (X,R,ℓ) where

X is a finite set of vertices,

$R \subseteq X \times X$ is an antireflextive relation,

$\ell : X \longrightarrow \mathcal{a}^*$ is a function ($\mathcal{a}^*$ is the set of all words of $\mathcal{a}$).

The way the words interact is given by a programming language. An instruction written in this reproductive language may somehow rearrange its closest neighborhood, and a sequence of such instructions may act on its environment in quite a complicated way. There are following types of instructions:

1. TAKE instruction written in a word $\mathcal{L}(x)$ causes that a certain final part of some word $\mathcal{L}(y)$ where $(x,y) \in R$ is taken and appended behind $\mathcal{L}(x)$.
2. GIVE instruction written in a word $\mathcal{L}(x)$ causes that a certain final part of $\mathcal{L}(x)$ is taken and appended behind some word $\mathcal{L}(y)$ where $(x,y) \in R$.
3. MOVE instruction written in a word $\mathcal{L}(x)$ causes that some edge $(x,y) \in R$ is erased from R and some new edge (x,z) originate, provided $(y,z) \in R$
4. GOTO instruction (conditioned or unconditioned) enables branching in the sequence of instructions. Conditions allow inquiries into the structure of environment.
5. ASSIGN instruction causes that an arithmetical variable takes a given value. The variables are used as parameters of other instructions.

In order that a word may distinguish between its neighboring words, all edges sourcing from a given vertex are numbered consecutively. This is done by a numbering function r: $: R \longrightarrow \omega$ where ω is a set of all integers and r has the following properties:

1. $\forall (x,y) \in R \quad r(x,y) \geq 1$
2. $r(x,y) = r(x,z) \Longrightarrow y = z$ (1)
3. $r(x,y) > 1 \Longrightarrow (\exists z \in X)((x,z) \in R \,\&\, r(x,z) + 1 = r(x,y))$

If we have a graph inhabited by words of $\mathcal{Q}$, some of them being programs in our reproductive language, we would like that all these programs were carried out concurrently. However, the result would not be deterministic and therefore we adopt the following conventions:

There is given an integer k, a starting vertex $x_0 \in X$ and a successor function $s: X \longrightarrow X$ such that $s^n(x) \neq x$ for any $x \in X$ and any $n < \mathrm{card}\,(X)$. (2)

We imagine, there is a compiler that changes the graph as follows.

First it carries out k instructions of $\mathcal{L}(x_0)$, then k instructions of $\mathcal{L}(s(x_0))$ etc. If a word contains no instruction, the compiler ignores it and goes to next vertex. Having run through all vertices $x \in X$, the compiler returns to x_0 and goes on where it has interrupted. Throughout such computation, the compiler must remember two partial functions $i: X \longrightarrow \omega$, $v: X \times \mathrm{Var} \longrightarrow \omega$ where Var is the set of all variables. The function i specifies the address of interruption; for any $x \in X$ the program $\mathcal{L}(x)$ was interrupted at i(x)-th character, v(x,y) specifies the value of variable y at vertex x.

The idea of these graphs changing in discrete time steps is taken from [1]. In fact our system is a very special case of systems cons idered there.

Definition

A biosystem is a system $\mathcal{B} = (X, \mathcal{L}, R, r, x_0, s)$, where X is a finite set of vertices

$\mathcal{L} : X \longrightarrow \mathcal{A}^*$ is a labeling function,

$R \subseteq X \times X$ is an antireflextive relation,

$r: R \longrightarrow \omega$ is a numbering function with properties (1)

$x_0 \in X$ is a starting vertex, and

$s: X \longrightarrow X$ is a successor function with properties (2)

A computational state of a biosystem is a system $\mathcal{C} = (X, \mathcal{L}, R, r, x_0, s, i, v, k_0)$, where $(X, \mathcal{L}, R, r, x_0, s)$ is a biosystem,

$i: X \longrightarrow \omega$, $v: X \times Var \longrightarrow \omega$ are partial functions,

Var is the set of all variables and k_0 is an integer.

Definition

For any integer $k \geq 1$ an operator S_k acting on computational states is defined as follows:

If $\mathcal{C} = (X, \mathcal{L}, R, r, x_0, s, i, v, k_0)$ is a computational state, then $S_k(\mathcal{C}) = (X, \mathcal{L}', R', r', x_0', s, i', v', k_0')$ where $\mathcal{L}', R', r', v'$ are obtained by carrying out the instruction of $\mathcal{L}(x_0)$ on address $i(x_0)$,

$i'(x_0)$ is the address of next instruction, $i'(y) = i(y)$ for $y \neq x$, $k_0' = k_0 + 1$ and $x_0' = x_0$ if $k_0 < k$ and $k_0' = 1$ and $x_0' = s(x_0)$ if $k_0 \geq k$.

Definition

The initial computational state of a biosystem $\mathcal{B}$ is $\mathcal{B}_0 = (X, \mathcal{L}, R, r, x_0, s, i_0, v_0, 1)$, where v_0 is everywhere undefined, and $i_0(x)$ is address of first instruction in $\mathcal{L}(x)$ or undefined if there is no such instruction.

Definition

Let $\mathcal{B}$ be a biosystem, k an integer. We say that the word at $x \in X$ reproduces itself in $\mathcal{B}$ and S_k if there is an integer n with $S_k^n(\mathcal{B}_0) = (X, \mathcal{L}_1, R_1, r_1, x_1, s, i_1, v_1, k_1)$

$S_k^{n+1}(\mathcal{B}_0) = (X, \mathcal{L}_2, R_2, r_2, x_2, s, i_2, v_2, k_2)$ and $y \in X$ with $\mathcal{L}(x) = \mathcal{L}_2(x) = \mathcal{L}_2(y)$, $\mathcal{L}_1(x) = \mathcal{L}(x)\,\mathcal{L}(x)$, $\mathcal{L}_1(y) = \wedge$, $\mathcal{L}(x)\,\mathcal{L}(x)$ being concatenation and $\wedge$ the empty word.

In the second part of the paper we consider special biosystems in which all but one vertices of the underlying graph are labeled by a single character or empty word. These biosystems model an organism situated in a nonliving environment. We are interested in the development of this biosystem till the point, the organism reproduces itself (if it reproduces at all). By introducting the concept of random environment, we may compute probabilities that a program reproduces itself.

Definition

Let $\mathcal{B}$ be any biosystem. A path (between x_1 and x_n of length n) is any sequence $x_1, x_2, .x_n$ of vertices such that $(\forall i < n)(x_i, x_{i+1}) \in R$

Definition

Let $P \in \mathcal{A}^*$ be any program, a_1, a_2, a_3 reals with $a_1 > 0$, $a_2 \geq 0$, $a_3 > 0$ and $a_1 + a_2 + a_3 = 1$ (such triples will be called types).

A random environment of P of type (a_1, a_2, a_3) is a random variable whose values are biosystems $\mathcal{B} = (X, \mathcal{L}, R, r, x_0, s)$ such that

1. $\mathcal{L}(x_0) = P$, for any $y \in X$ $y \neq x_0 \Rightarrow \mathcal{L}(y) \in \mathcal{A} \cup \{\wedge\}$ and for any $\alpha \in \mathcal{A} \cup \{\wedge\}$ prob $[\mathcal{L}(y) = \alpha] = 1/A$, where $A = \text{card}(\mathcal{A}) + 1$

2. There is exactly one $y \in X$ with $(x_0, y) \in R$, for any two vertices there is at most one path between them, for any $y \in X$ $y \neq x_0 \Rightarrow$ prob [there are exactly $i - 1$ edges sourcing from y] $= a_i$ ($a_i = 0$ for $i > 3$)

3. all these probabilities are independent.

Definition

For any type (a_1, a_2, a_3) the reproductive probability p of a program P is the probability that P reproduces itself in a random environment of type (a_1, a_2, a_3).

Definition

For any program P, its depth is the least integer k such that P inspects only such vertices $y \in X$ that there is a path of length at most k between x_0 and y. (The depth of P may be deciphered from its syntax.)

Theorem

For any type (a_1, a_2, a_3) and any integer $k \geq 2$ there is a positive real number $q_k < 1$ such that the reproductive probability of any program of depth k and length n is less or equal to q_k^{n-k} .

Consequence

For any type (a_1,a_2,a_3) and any integer k, the set of reproductive probabilities of all programs of depth k has a maximal element (denoted m_k).

Programs with greater depth have greater length, and we do not know yet whether their reproductive probabilities tend to 1 owing to increase in q_k or to 0 owing to increase in length. This question is answered by next theorem.

Definition

For a given type (a_1,a_2,a_3) let r_k = prob [in a random environment of type (a_1,a_2,a_3) there is no path of length k].

There is $r_1 = 0$ and $r_{k+1} = a_1 + a_2 r_k + a_3 r_k^2$ and $\lim r_k =$ $= \min(1,a_1/a_3)$. Surely, if P has length n then its reproductive probability $p \leq 1 - r_n$

Theorem

There is a sequence of programs P_k of depth k such that for any type (a_1,a_2,a_3), if p_k is the reproductive probability of P_k then

$$\lim_{k\to\infty} p_k = 1 - \lim_{k\to\infty} r_k = \max(0, 1 - a_1/a_3)$$

Consequence

The set $\{m_k\}_{k\geq 2}$ has a maximal element if and only if there is $k\geq 2$ with $m_k \geq \max(0, 1 - a_1/a_3)$

In the last part of the paper we consider (without any mathematical treatment) biosystems with two or more "living" programs. An example of program that purposefully competes with its neighbours, is exhibited.

References

[1] V. Rajlich: Dynamics of discrete systems and pattern reproduction, JCSS Vol 11,No 2, October 1975, pp 186-202