Intern. J. Computer Math. 1979, Section A, Vol. 7, pp. 119–128

Printed in Great Britain

Recursion Theory on Processes

JAN BERGSTRA

Institute of Applied Mathematics and Computer Science,
University of Leiden, Wassenaarseweg 80, Leiden, The Netherlands

A general definition of a process is given. Recursive processes and relative recursion for processes are defined and the degree theory of processes is studied. We construct a process of minimal degree.

KEY WORDS: Process, recursion.

C.R. CATEGORIES: 5.22, 5.27.

1. INTRODUCTION AND PRELIMINARIES

1.1 Introduction

In this paper we study processes from the point of view of recursion theory. Our purpose is twofold:

i) analysis of processes in general,

ii) to introduce a dynamical aspect in recursion theory.

It is our opinion that recursion theory is of a very static character. To make it more applicable to computer science it must deal with more dynamical situations. Considering processes rather than functions is one way to introduce a dynamical element.

A process P is conceived as an entity which communicates with the outside world. It has an initial state P_ε and it answers every question q in L_Q (question language) with an answer a in L_A (answer language). Answers depend on the history of the process only.

We can imagine applications of a general theory of processes in the following fields:

i) Specification and semantics of datatypes (see [2]) and programming languages.

ii) Theory of learning; a general theory of the learning properties of recursive processes seems to be of interest. Further a theory of recursive

processes might provide tools for the abstract description of artificial intelligence results in which often some sort of learning occurs.

iii) Generalities concerning scheduling and deadlock problems.

It seems that our recursion theory on processes is new. A treatment of processes as transducers from infinite question sequences to infinite answer sequences in terms of λ-calculus is given by Milnor in [3].

1.2 Definition of (recursive) processes

Names of processes will be $P, P^1, P^2, \ldots, Q, R, S \ldots$. There is a problem in what exactly is a process. Suppose P is asked question q and has given answer a. How to describe the new situation? P has become entirely different. Clearly P is just a name and at any moment a function (ext) must assign to P its extensional meaning (extension, graph, stage) ext (P).

1.2.1

We *define* the collection PR of *process extensions* (graphs) as follows:

$$PR = L_Q^* \to L_A \text{ (for given } L_Q \text{ and } L_A).$$

Let $F = \text{ext}(P)$. Then the answer of P on question q is $F(q)$ and the new value of ext (P) is $\lambda\sigma \in L_Q^* \cdot F(q * \sigma)$.

Conventions P_ε usually denotes the initial extension of P, P_σ denotes the extension of P after questions $\sigma_1, \ldots, \sigma_n$ $(n = \ln(\sigma))$ have been answered.

As we will not consider any details of L_Q and L_A we identify both of them with ω, the set of natural numbers, in this paper. This can be done without loss of generality. Instead of L_Q^* we have the sequence numbers $\sigma = \langle \sigma_0, \ldots, \sigma_{n-1} \rangle$ from now on.

1.2.2

DEFINITION P is a recursive process if ext (P) is a recursive function.

1.2.3

An example Infinitary union find process. At any stage the sets $V_i i \in I (I \subseteq \omega)$ are disjoint finite subsets of ω such that $\bigcup_{i \in I} V_i = \omega$. At the initial stage $V_i = \{i\}$ for all i. Further, anytime i is itself the smallest element of V_i for all $i \in I$. We look for a process P which can keep track of the effect of union instructions on this collection of sets in such a way that it is always able to correctly answer find questions (FIND(j)). The union instructions (UNION(i, j)) are also seen as questions. FIND(j) asks for the

$i \in I$ such that $j \in V_i$. UNION(i, j) has the following effect: I becomes $(I - \{\max(i, j)\}) \cup \{\min(i, j)\}$, $V_{\min(i,j)}$ becomes $V_i \cup V_j$. The answer on UNION-questions is not specified. It is clear that we may choose for P a recursive process. If, however, we start with another initial situation with V_i not uniformly recursive in i we need a nonrecursive process to keep track of the unions.

Remark A process P has according to our definition at any stage a finite history starting with its initial graph P_ε. It is straightforward to introduce processes with an infinite history as sequences $\ldots P_{-2}, P_{-1}, P_\varepsilon$ of graphs such that $P_{-(i+1)}$ transforms to P_{-i} after answering question q_i with answer a_i. It is then possible to extend "relative recursion" consistently to processes with an infinite history.

2. RECURSION ON PROCESSES

The aim of this section is to define a satisfactory notion of relative recursion for processes.

2.1

We start with a definition of computations which take processes and natural numbers as an input and have natural numbers as an output (if any). We use the notation of Kleene brackets. The details, however, are more like in [1]. We will define a relation $\{e\}(\mathbf{m}, \mathbf{P}) \cong n$ which expresses: the computation with index e and arguments $\mathbf{m}, \mathbf{P}$ leads to value n. More precisely we define a collection C of triples

$$t = \langle \mathbf{A}, [e, \mathbf{m}, \mathbf{P}, n], \mathbf{B} \rangle$$

which code computations in the following way: If $\text{ext}(P^i) = A^i (i \leqq l)$ then the computation $\{e\}(\mathbf{m}, \mathbf{P})$ leads to value n and after the computation $\text{ext}(P^i) = B^i (i \leqq l)$.

The set C is given by an inductive definition with four clauses each corresponding to one of the elementary computation steps. The case distinction is according to the structure of e. As usual we present the induction in terms of schemes. We use schemes $R_1, \ldots, R_4$. To distinguish the cases there are four predicates T, S, A, R which we will now define.

$T(e) \equiv e = \langle 1, p \rangle$ for some p.
e is index of a terminal computation (i.e. a computation without subcomputations).

$S(e) \equiv e = \langle 2, p, q \rangle$ for some p, q.

e codes the substitution of two computations.

$A(e) \equiv e = \langle 3, p, q\rangle$ for some p, q.

e codes the application of a process argument on a numerical argument (question).

$R(e) \equiv e = \langle 4, p\rangle$ for some p.

e is a reflection index.

We will now describe the schemes:

let $$t = \langle \mathbf{A}, [e, \mathbf{m}, \mathbf{P}, n], \mathbf{B}\rangle,$$

R_1; if: $T(e)$, $e = \langle 1, p\rangle$, $A^i = B^i (i \leqq l)$ and $[p](\langle \mathbf{m}\rangle) = n$ then $t \in C$.
Here $[p]$ is the p-th unary primitive recursive function in some indexing of the pr.rec. functions.

Remarks As this computation does not use the process arguments these remain unaffected. For purposes of complexity theory it might be more useful not to introduce all primitive recursive computations as atomic steps.

Informal notation $\{e\}(\mathbf{m}, \mathbf{P}) \cong [p](\langle \mathbf{m}\rangle)$.

R_2; if: $S(e)$, $e = \langle 2, p, q\rangle$,
$\langle \mathbf{A}, [p, \mathbf{m}, \mathbf{P}, k], \mathbf{D}\rangle \in C$ for some k, D
and $\langle \mathbf{D}, [q, k, \mathbf{m}, \mathbf{P}, n], \mathbf{B}\rangle \in C$
then $t \in C$.
Informal notation $\{e\}(\mathbf{m}, \mathbf{P}) \cong \{q\}(\{p\}(\mathbf{m}, \mathbf{P}), \mathbf{m}, \mathbf{P})$.

R_3; if: $A(e)$, $e = \langle 3, p, q\rangle$
$A^p(m_q) = n$, $B^p = \lambda\sigma \cdot A^p(\langle q\rangle * \sigma)$ and for $i \neq p$
$B^i = A^i$
then $t \in C$.
Informal notation $\{e\}(\mathbf{m}, \mathbf{P}) \cong P^p(m_q)$.

R_4; if: $R(e)$, $e = \langle 4, p\rangle$, $\langle \mathbf{A}, [m_0, m_{(p)_1}, \ldots, m_{(p)_l}, \mathbf{P}, n], \mathbf{B}\rangle \in C$
then $t \in C$ (here $p = \langle (p)_1, \ldots, (p)_l\rangle$).
Informal notation $\{e\}(\mathbf{m}, \mathbf{P}) \cong \{m_0\}(\mathbf{m}^p, \mathbf{P})$.

C is the minimal set which satisfies the closure properties $R_1, \ldots, R_4$. Informally we write $\{e\}(\mathbf{m}, \mathbf{P}) \cong n$ if $\mathrm{ext}(P^i) = A^i$ and for some B^i $\langle \mathbf{A}, [e, \mathbf{m}, \mathbf{P}, n], \mathbf{B}\rangle \in C$.

2.2

Relative recursion. We see a process now as a pair of machine and state. The machine computes for every question the answer and the next state. For machine (index) e, initial state s_e and processes $\mathbf{P}$ we define the (partial) process Q, informally denoted by $\lambda \cdot \{e\}^{\mathbf{P}}(s_e, \cdot)$, using induction on

the length of question sequences as follows:

$$Q_\sigma(q) = (\{e\}(s_\sigma, q, \mathbf{P}))_0$$
$$s_\sigma * \langle q \rangle = (\{e\}(s_\sigma, q, \mathbf{P}))_1.$$

DEFINITION Q is recursive in $\mathbf{P}$ if for some e, s_ε in ω

$$Q = \lambda \cdot \{e\}^{\mathbf{P}}(s_\varepsilon, \cdot).$$

2.3

1) $\leqq$ is transitive in the following sense: $P \leqq P^1 \ldots P^k$ and $P^i \leqq P^{i,1} \ldots P^{i,j_i} (i \leqq k)$ imply $P \leqq P^{1,1} \ldots P^{1,j_1} \ldots P^{k,1} \ldots P^{k,jk}$.

2) It is not the case that $P \leqq R$, R implies $P \geqq R$.

3) If we consider partial processes it is possible to prove versions of the first and second recursion theorem.

4) We may allow the definition of new argument processes, during the computation of a process, from a relative index and some other processes, without changing $\leqq$, as long as we take into account the side effects that use of a defined process has on the processes from which it has been defined.

5) If for all σ $P_\sigma = P_{\varepsilon|}$ then P is just a function. In that case recursion relative to P as a process is equivalent to recursion relative to P_ε as a function.

6) For processes P and Q we define a kind of join $P \wedge Q$ informally as follows: on a question $\langle 0, q \rangle P \wedge Q$ answers $P(q)$, on $\langle 1, q \rangle$ it answers $Q(q)$, and on $\langle n+2, q \rangle$ it answers 0. Clearly $P, Q \leqq P \wedge Q \leqq P, Q$. (Unfortunately $\wedge$ is not the join w.r.t. $\leqq$.)

7) We assume Church's thesis for this (relative) recursion and will work with intuitive descriptions of algorithms rather than formal descriptions.

2.4

DEFINITION $P \equiv Q$ if $P \leqq Q$ and $Q \leqq P$.

$\leqq$ induces a partial ordering on $PR/\equiv$.

$\langle PR/\equiv, \leqq, 0 \rangle$ is the structure of degrees of processes. (0 is the degree of recursive processes.)

Of course it seems interesting to introduce (discrete) time as an explicit extra parameter of processes. This enables us to study parallelism and to

give the processes an own life. It is clearly not necessary to have every response of a process triggered by a question in this case. A problem, however, is inescapable in defining recursion. The meaning of relative recursion will depend strongly on the details of its definition. So it is to be expected that transitivity fails and so on. However, if we succeed in dividing out a suitable equivalence relation on the processes in order to identify processes which use a "similar" amount of time for "the same" activities, we may be able to define a satisfactory recursion theory on these equivalence classes.

3. DEGREES

In this section we study some properties of $\langle PR/\equiv, \leqq, 0\rangle$. We mainly consider the role of constant processes and density questions.

3.1

DEFINITION A process P is constant if for all σ $P_\varepsilon = P_\sigma$. A degree is constant if it contains a constant process.

3.2

THEOREM *For every degree d there is a unique constant degree c above d.*

Proof Let $P \in d$, with initial graph P_ε. We define Q as follows:

$$Q_\varepsilon(\langle s_1\rangle * \ldots * \langle s_k\rangle) = P_\varepsilon(\langle s_{l+1}-1\rangle * \ldots * \langle s_k - 1\rangle)$$

where l is maximal $\leqq k$ such that $s_l = 0$. Q is as strong as P but allows for an initialisation.

In fact asking question 0 brings it back to its initial stage. Clearly $P \leqq Q$. To see that Q is of constant degree note that it is recursively equivalent with P_ε as a function. Now suppose that R is constant and $P \leqq R$. We show $Q \leqq R$ and then we take c to be the degree of Q. The algorithm for Q is as follows (with $P = \lambda \cdot \{e\}^R(s_\varepsilon, \cdot)$).

$$Q(q) = \begin{cases} \textit{if } q = 0 \quad \textit{then} & \text{answer: } 0 \\ & \text{new state: } s_\varepsilon \\ \textit{if } q = n+1 \quad \textit{then} & \text{answer: } (\{e\}(s, R))_0 \\ & \text{new state: } (\{e\}(s, R))_1 \end{cases}$$

□

3.3

DEFINITION $P \leqq_{\cup} R$ (p is uniformly recursive in R) if for some e, s_ε the following holds.

$$P = \lambda \cdot \{e\}^{R_\sigma}(s_\varepsilon, \cdot) \text{ for each } \sigma.$$

Motivation The user of a process prefers not to depend on the precise stage of a process.

However, one easily shows:

3.4

THEOREM *If* $P \leqq_{\cup} R$ *then* $Q \leqq R$ *where* Q *is as in the proof of* 3.2.

Therefore we conclude that $PR/\equiv_{\cup}$ is not interesting. ($\equiv_{\cup}$ is not even reflexive.) This observation reveals a difficulty in finding applications for recursion on processes.

3.5

THEOREM *If* a *and* b *are constant degrees with* $a < b$ *then for some* c (*not necessarily constant*) $a < c < b$.

Proof Let $A \equiv f_a$, $B \equiv f_b$ (f_a and $f_b \in \omega \to \omega$).
Let P^B be defined by

$$P^B_\varepsilon(\sigma) = \textit{if } \sigma = \langle s \rangle \textit{ then } f_b(s) \textit{ else } 0 \textit{ fi}.$$

Let $C = A \wedge P^B$ and take c the degree of C.
Clearly $A \leqq C \leqq B$.
Suppose $A \geqq C$ then $A_\varepsilon \geqq C_\varepsilon$ as functions, hence $f_a \geqq A_\varepsilon \geqq c_\varepsilon \geqq f_b$. But $f_a < f_b$, contradiction.
On the other hand if $C \geqq B$ then we derive a contradiction as follows:

First prove: $f \leqq C \Rightarrow f \leqq A$ (the idea is that in the computation of the function f only once nontrivial information concerning f_b can be used; in fact $f \leqq A$ is not true in a uniform way). Now $C \geqq B$ implies $f_b \leqq C$, so $f_b \leqq A \leqq f_a$, contradiction. □

COROLLARY *There is a non-constant degree.*

Proof Consider A, B, C as in the previous proof. If $C \equiv g$ then $g \leqq C$ hence $g \leqq A$ hence $C \leqq A$. So C is not of a constant degree. □

3.6

THEOREM *There exists a process of minimal nonzero degree.*

Proof For a function f we define P^f by $P^f_\varepsilon(\sigma) = \textit{if } \sigma = \langle s \rangle \textit{ then } f(s) \textit{ else } 0 \textit{ fi}$.

Clearly $0 \leqq P^f \leqq f$. P^f is going to be of minimal (nonzero) degree for suitable f. To ensure $P^f > 0$ we make f nonrecursive ($P^f \leqq 0 \to P^f_\varepsilon \in 0 \to f \leqq 0$).

Suppose $Q < P^f$. Look at a computation $\lambda \cdot \{e\}^{P^f}(s_\varepsilon, \cdot)$ for Q from P^f. We replace it by a computation $Q_\varepsilon = \lambda\sigma \cdot \{a\}^f(\sigma)$ where f may be used at most once for every computation on a question sequence σ. (After this first use the value $f(n)$ is always considered zero.) We will define f such that all its values are either zero or one.

Let R_a be the following set of quadruples:

$$R_a = \{\langle \sigma, i, k, l \rangle \mid \forall g \in 2^\omega \{a\}^g(\sigma) = \textit{if } g(i) = 0 \textit{ then } k \textit{ else } l \textit{ fi}\}$$

R_a is recursively enumerable, one need only experiment with $g = \lambda x \cdot 0$ and $g = \lambda x \cdot 1$.

Let $R_{a,I} = \{i \in \omega \mid \exists t \in R, \exists \sigma, k, l \quad t = \langle \sigma, i, k, l \rangle\}$.

3.6.1

Suppose $R_{a,I}$ is finite then information concerning finitely many values $f(i)$ is sufficient to compute all values of $\{a\}^f(\sigma)$, hence Q is recursive. So assume that $R_{a,I}$ is infinite. We find that it has an infinite recursive subset $R^1_{a,I}$.

Let $f_U(x) = \textit{if } x \in V \textit{ then } f(x) \textit{ else } 0 \textit{ fi}$ for $V \subseteq \omega$.

3.6.2

LEMMA $P^{fs} \leqq Q$ *for recursive* $S \subseteq R_{a,I}$.

Proof We must only show that $P^{fs}(\langle s \rangle) = f(s)$ can be computed for $s \in \omega$. Given s decide $s \in S$. If not then 0, if so then look for σ, k and l such that $\langle \sigma, s, k, l \rangle \in R_a$. Then test $Q(\sigma)$. Now we know:

$$f(s) = \textit{if } Q(\sigma) = k \textit{ then } 0 \textit{ else } 1 \textit{ fi}. \qquad \square$$

We call $R_{a,I}$ the domain of relevance of f w.r.t. Q. Clearly, if $\lambda \cdot \{e\}^{P^g}(s_\varepsilon, \cdot)$ and $\lambda \cdot \{e\}^{P^h}(s_\varepsilon, \cdot)$ are both total and $g \cap R_{a,I} = h \cap R_{a,I} = f \cap R_{a,I}$ then they are equal. So we have shown that given e, s_ε either $\lambda \cdot \{e\}^{P^f}(s_\varepsilon, \cdot)$ is recursive or for every recursive subset S of the domain of relevance in question, $P^{fs} \leqq q (= \lambda \cdot \{e\}^{P^f}(s_\varepsilon, \cdot))$.

3.6.3

LEMMA *There exists a nonrecursive f such that for every r.e. set V one of the following is true:*

i) *f_V is almost everywhere zero,*

ii) *there is an infinite recursive subset S of V such that $f_{\bar{S}}$ is zero a.e.*

Proof Let V_i be an enumeration of the r.e. sets. We define f by means of an infinite sequence of recursive sets $R_i (i \in \omega)$ which is inductively defined in a straightforward manner such that the following conditions are satisfied:

i) $R_i \supseteq R_{i+1}$ for all i and R_i is infinite for all i;

ii) $f_{\omega \setminus R_0}$ and $f_{R_i \setminus R_{i+1}}$ are zero a.e. (for all i);

iii) f is non-recursive;

iv) if $V_i \cap R_i$ is finite then $R_{i+1} \subseteq V_i$;

v) if $V_i \cap R_i$ is finite then $R_{i+1} \cap V_i = \emptyset$.

It is easy to see that any f satisfying (i)...(v) has the required properties. □

3.6.4

Now we prove 3.6. Let f be as in 3.6.3. We prove that P^f is minimal. Clearly, $P^f > 0$. Consider $Q = \lambda \cdot \{e\}^{P^f}(s_\varepsilon, \cdot)$. Suppose $Q \not\equiv 0$. Let $R_{a,I}$ be the domain of relevance in this case. (In view of 3.6.1 $R_{a,I}$ is infinite.) There are now two cases (by 3.6.3):

i) $f_{R_{a,I}}$ is zero a.e. In this case one easily proves that Q is recursive in spite of the assumptions. So this case leads to a contradiction;

ii) $f_{\bar{S}}$ is zero a.e. for some infinite recursive $S \subseteq R$ (and $R^1_{a,I}$ is an example of such a set S). We take such an S and show:

3.6.4.1

PROPOSITION *$P^f \leqq P^{fs}$. This is easy, to compute $P^f(\langle s \rangle) = f(s)$ first decide if $s \in S$. If so then use P^{fs}, if not a recursive function gives the answer as $f_{\bar{S}}$ is zero a.e.*

3.6.4.2

PROPOSITION $Q \leqq P^{fs}$.

Proof Let σ be given. To compute $Q_\varepsilon(\sigma) = \{e\}^{P^f}(s_0, *)(\sigma) = \{a\}^f(\sigma)$ using only one value of f_S one proceeds as follows: Decide if the computation

requires information concerning f. If not, compute the answer. If so let l be the argument for which information is required. Now proceed as in 3.6.4.1.

Taking this together we find (for case ii)):

$$P^f \leqq P^{f\,s} \leqq Q \leqq P^{f\,s} \leqq P^f \text{ hence } P^f \equiv Q.$$
$$\text{(3.6.4.1) (3.6.4.2) (3.6.2) (easy)}$$

This concludes the proof of 3.6.

References

[1] J. A. Bergstra, Computability and continuity in finite types, Ph.D. thesis, Utrecht, 1976.
[2] J. A. Bergstra, Nondeterministic datatypes (forthcoming).
[3] R. Milner, Processes, a mathematical model of computing agents. (Logic Colloquium, 1973, North-Holland), 157–173.