A PROCESS CREATION MECHANISM IN PROCESS ALGEBRA

*J.A. Bergstra, Department of Computer Science,
University of Amsterdam.
Department of Philosophy,
University of Utrecht.*

Logic Group
Preprint Series
No. 2

Department of Philosophy

University of Utrecht

A PROCESS CREATION MECHANISM IN PROCESS ALGEBRA

J.A. Bergstra, Department of Computer Science,
University of Amsterdam.
Department of Philosophy,
University of Utrecht.

This research was sponsored in part
by ESPRIT project METEOR (nr. 432).

September 1985.

University of Utrecht,
Department of Philosophy,
Heidelberglaan 2,
3584 CS Utrecht,
The Netherlands.

A PROCESS CREATION MECHANISM IN PROCESS ALGEBRA

J.A. Bergstra, Department of Computer Science,
University of Amsterdam.
Department of Philosophy,
University of Utrecht.

ABSTRACT

We introduce an encapsulation operator E_φ that provides process algebra with a process creation mechanism. Several simple examples are considered. It is shown that E_φ does not extend the defining power of the system 'ACP with guarded recursion'.

1. Introduction

1.1 Within Esprit Project Meteor we work on process algebra because that fits the description of Meteor's task 2: specification techniques. An overall goal of the theory part of Meteor is to obtain a uniform semantic description of a wide range of phenomena connected with programming. We hope that process algebra can lead to a uniform description of many issues in concurrency.

In order to make progress in this direction we try to incorporate more and more features in process algebra. In many cases a new feature requires new (additional) syntax and more equations. The core system ACP, see [4, 5, 6] describes asynchronous cooperation with synchronous communication.

On top of ACP various features can be added, for instance: asynchronous communication [7], cooperation in the presence of shared data [1], broadcasting [3], interrupts [2].

This note adds process creation to the features that are compatible with process algebra.

For historical remarks and relations with previous literature we refer to [4].

1.2 We start on basis of the axiom system ACP, which is supposed to be known to the reader.

We assume the presence of a finite set of data D and introduce for each $d \in D$ an action cr(d). cr(d) stands for: create a process on basis of initial information d. (Let $cr(D)=\{cr(d)\,|\,d\in D\}$).

Let φ be a mapping that assigns to each $d \in D$ a process $\varphi(d)$. Then the operator E_φ (process creation encapsulation w.r.t. φ) is defined by the following equations. We assume that always $cr(d)|a=\delta$ and never $a|b=cr(d)$.

$$\boxed{\begin{array}{l}
E_\varphi(\delta)=\delta \\
E_\varphi(a)=a.\delta \text{ if } a\notin cr(D) \\
E_\varphi(a.X)= a.E_\varphi(X) \text{ if } a\notin cr(D) \\
E_\varphi(cr(d))=\overline{cr}(d).E_\varphi(\varphi(d)) \\
E_\varphi(cr(d).X)=\overline{cr}(d).E_\varphi(\varphi(d)\|X) \\
E_\varphi(x+y)=E_\varphi(x)+E_\varphi(y)
\end{array}}$$

here $\overline{cr}(d)$ is a new atom which indicates that process creation has taken place ($\varphi(d)$ is "born").

As usual it is the case that on all finite terms E_φ can be eliminated (provided $\varphi(d)$ contains no cr(d) actions). In any case one can compute for each n the n-th projection $\Pi_n(t)$ of a term with E_φ as a term without E_φ by applying the equations as rewrite rules from left to right.

2. Very small examples

In this section we provide several examples that should support the claim that E_φ properly describes process creation on top of ACP. It should be noted that we have no abstraction present, in terms of [1] we are dealing with concrete process algebra.

2.1 $D=\{d\}$ $\varphi(d)=cr(d)$

Let $P=E_\varphi(cr(d))$, then $p=\overline{cr}(d).E_\varphi(\varphi(d))=\overline{cr}(d).E_\varphi(cr(d))=\overline{cr}(d).p$.

It follows that E_φ involves recursion already under the simplest conditions.

We assume that we will always use guarded recursive specifications, and have a semantics in the standard model of graphs modulo bisimulation. Then one may use the approximation induction principle (see [1]) AIP:

$$\frac{\text{for all } n\ \Pi_n(X)=\Pi_n(Y)}{X=Y}$$

Using AIP one can prove that in the absence of communication ($a|b=\delta$ for all a,b) the following holds:

$$(*)\quad E_\varphi(X\|Y)=E_\varphi(X)\|E_\varphi(Y)$$

This leads to the second example:

2.2 Let $D=\{d\}$, $\varphi(d)=a.cr(d)\|b.cr(d)$

$a|b=\delta$ and $p=E_\varphi(cr(d))$.

Then $P=\overline{cr}(d).E_\varphi(a.cr(d)\|b.cr(d))=$(using $*$)

$\overline{cr}(d).E_\varphi(a.cr(d))\|E_\varphi(b.cr(d))=$

$\overline{cr}(d).(a.E_\varphi(cr(d))\|b.E_\varphi(cr(d))=$

$\overline{cr}(d).(ap\|bp)$.

2.3 Let $D=\{d\}$, $\varphi(d)=a.(cr(d)\|cr(d))+b$

let $a|b=\delta$ and put $p=E_\varphi(cr(d))$.

Now $p=E_\varphi(cr(d))=\overline{cr}(d).E_\varphi(a.(cr(d)\|cr(d))+b)=$

$\overline{cr}(d).aE_\varphi(cr(d)\|cr(d))+b\delta=\overline{cr}(d).a(p\|p)+b\delta$

(again using $*$).

We observe that the "population of p's" may either grow or disappear.

3. Small but genuine examples

3.1 A population of animals. Let D be a finite set of genetic codes provided with a mixing operation $*:D\times D\rightarrow D$ and a predicate $F(\text{female})\subseteq D$.

Let for $a \in D$:

$$p^a = (hunt(a)+sleep(a)+eat(a)+idle(a)).p^a + end(a)$$

if $a \in F$: $q^a = \sum_{b \in D} pair(a,b).(create(a*b).q^a + end(a)) + end(a)$

if $a \notin F$: $q^a = \sum_{b \in D} pair(a,b).q^a + end(a)$.

Take the following communication function:

$$end(d) \mid end(d) = \overline{end}(d)$$

$$pair(a,b) \mid pair(b,a) = \overline{pair}$$

(all other communications δ).

Let $H_0 = \{end(d) \mid d \in D\}$

$H_1 = \{pair(a,b) \mid a,b \in D\}$.

Then define $\varphi(a) = \partial_{H_0}(p^a \| q^a)$

and $S = \partial_{H_1}(E_\varphi(\varphi(a) \| \varphi(b)))$

S describes a population of animals $\varphi(d)$, starting with two individuals. Each animal can hunt, sleep, eat, idle, pair, create (when female) and end (its life).

The population can develop in many different ways, in particular it can die out. Very simple observations can be made on this system. For instance if both a and b are male (i.e. $\notin F$) then no create action will take place.

3.2 The bag

Let $B = \sum_{d \in D} read(d).cr(d).B$

$\varphi(d) = write(d)$.

Consider $B^* = E_\varphi(B)$

then $B^* = E_\varphi(\sum_{d \in D}(read(d).cr(d).B)) =$

$$\sum_{d \in D} read(d).\overline{cr}(d).E_\varphi(\varphi(d) \| B) =$$

$$\sum_{d \in D} read(d).\overline{cr}(d).E_\varphi(write(d) \| B) =$$

$$\sum_{d \in D} read(d).\overline{cr}(d).(write(d) \| B^*)$$

Here we use again the fact that in the absence of communication E_φ distributes over $\|$.

If we now use abstraction and substitute τ for $\overline{cr}(d)$ then we obtain (with $I = \{\overline{cr}(d) \mid d \in D\}$ and τ_I as in [5]) the equation for a bag over D,

$$\tau_I(B^*) = \sum_{d \in D} read(d).(write(d) \| \tau_I(B^*))$$

This equation was discussed in several earlier papers, for instance [6].

The above calculation shows the intuition behind the equation for $\tau_I(B^*)$: the read(d) action creates the option to write (d).

3.3 A sieve of Eratostenes

We will write a program that generates all prime numbers in $\overline{N}=[1,\ldots,N]$ in increasing order. The program is called SIEVE, all its internal steps and communications will be t, and therefore it is claimed (but not proved) that $\tau_{\{t\}}(\mathrm{SIEVE})=\mathrm{write}(2),\mathrm{write}(3)\ldots\ \mathrm{write}(q).S$ where 2,3... q enumerates the primes in $\overline{N}$ in increasing order.

The design of SIEVE is derived from an example of P. America from PRLE, written to illustrate the object oriented programming language POOL.

3.3.1 Alphabet of actions A

δ: deadlock

t: internal step

for all $i\in\overline{N}$

write (i) output i

for all $i,j\in\overline{N}$

send i(j) send j through port i

read i(j) read j through port i

create (i,j): create a new process from data i,j

we write $\overline{\mathrm{create}}\ (i,j)=t$

(thus $D=\overline{N}\mathrm{x}\overline{N}$).

$$t(i=0 \bmod j)=\begin{cases} t \text{ if } i=0 \bmod j. \\ \delta \text{ otherwise.}\end{cases}$$

$$t(i\neq 0 \bmod j)=\begin{cases} t \text{ if } i\neq 0 \bmod j. \\ \delta \text{ otherwise.}\end{cases}$$

communication function:

send i(j) | read i(j) = t

all other communications are δ.

Counting the actions we find $\#(A)=5N^2+N+2$

3.3.2 Construction of SIEVE

We have $\mathrm{SIEVE}=\partial_H E_\varphi(S_1)$ where H,φ and S_1 are given below.

$H=\{\mathrm{send}\ i(j),\mathrm{read}\ i(j)\mid i,j\in\overline{N}\}$

$S_1=\mathrm{create}(1,2).\mathrm{send}1(3).\mathrm{send}1(4)\ldots\ \mathrm{send}1(N)$

thus S_1 creates a process for the (first) prime 2 and then sends all numbers in [3,N] in increasing order through port 1. (These messages are going to be received by φ(1,2).

$\varphi(i,j)=S_j^i$ with

S_j^i=write (j).
$\sum_{z\in \underline{N}}$ read i(z) $[t(z=0 \bmod j).S_j^i+t(z\neq 0 \bmod j).\text{create}(j,2).R_j^i]$

$R_j^i=\sum_{z\in \underline{N}}$ read i(z) $[t(z=0 \bmod j).R_j^i+t(z\neq 0 \bmod j).\text{send } j(z).R_j^i]$

The explanation of S_j^i is as follows:
S_j^i will be created as soon as a new prime j is found by S_i^l (here l,i,j are consecutive primes). The first task of S_j^i is to output j, then it receives numbers and checks all of these on being 0 mod j. The first z≠0 mod j must be a prime (it has survived the entire pipeline from S_1 to S_2^1 to S_3^2 to S_5^3... till S_j^i). For this z a new process (S_z^j) is created. Thereafter S_j^i restricts itself to filtering out all numbers in the pipeline that are divisible by j and transmitting the others to S_z^j.

4. E_φ can already be defined in ACP

We assume a situation where the E_φ operator is not explicitly mentioned in the definition of φ. All foregoing examples are of that nature.

We will then show how to eliminate E_φ in favour of synchronous communication. For each d∈D an action create*(d) is introduced and communication works as follows:

$\text{create}(d)\,|\,\text{create*}(d)=\overline{\text{create}}(d)$

(the create-actions are not involved in any other proper communications).

Now define K_φ *as follows:*

$K_\varphi=\sum_{d\in D}\text{create*}(d).(K_\varphi \| \varphi(d))$

Let $H=\{\text{create}(d),\text{create*}(d)\,|\,d\in D\}$.
Suppose that p does not contain E_φ.
Then:

$$\boxed{E_\varphi(p)=\partial_H(K_\varphi \| p)}$$

Note that K_φ does not involve E_φ anymore.
We support this identity by showing that the operator $p\to\partial_H(K_\varphi \| p)$ satisfies the defining equations of E_φ.
On appropriate models, like the standard graph model modulo bisimulation it can be shown that this type of *functional recursion* has a unique solution indeed.

$p=\delta \quad E_\varphi(p)=\delta$
$\partial_H(K_\varphi \| p)=\partial_H(K_\varphi.\delta)=\delta=E_\varphi(p)$

$p=X+Y \quad E_\varphi(p)=E_\varphi(X)+E_\varphi(Y)$

$$\begin{aligned}&\partial_H(K_\varphi \| p)=\partial_H(K_\varphi \mathbin{⌊\!⌊} p)+\partial_H(K_\varphi | p)+\partial_H(p \mathbin{⌊\!⌊} K_\varphi)=\\ &\delta+\partial_H(K_\varphi | X)+\partial_H(K_\varphi | Y)+\partial_H(X \mathbin{⌊\!⌊} K_\varphi)+\partial_H(Y \mathbin{⌊\!⌊} K_\varphi)=\\ &\partial_H(K_\varphi \mathbin{⌊\!⌊} X)+\partial_H(K_\varphi | X)+\partial_H(X \mathbin{⌊\!⌊} K_\varphi)+\partial_H(K_\varphi \mathbin{⌊\!⌊} Y)+\partial_H(K_\varphi | Y)+\partial_H(Y \mathbin{⌊\!⌊} K_\varphi)=\\ &\partial_H(K_\varphi \| X)+\partial_H(K_\varphi \| Y)=\\ &E_\varphi(X)+E_\varphi(Y)=p\end{aligned}$$

then there are the cases

$$\left.\begin{array}{l} p=a \\ \quad aX \end{array}\right\} \text{ a not of the form create(d)}$$

create(d)

create(d).X

We consider the last case only the others being similar or simpler.

p=create(d).X

$$E_\varphi(p)=\overline{\text{create}}(d).E_\varphi(X \| \varphi(d))$$

$$\partial_H(K_\varphi \| p)=\partial_H(\sum_{a\in D} \text{create}^*(a).(K_\varphi \| \varphi(d)) \| \text{create}(d).X)$$

$$=\overline{\text{create}}(d).\partial_H((K_\varphi \| \varphi(d)) \| X)$$

$$=\overline{\text{create}}(d).\partial_H(K_\varphi \| (\varphi(d) \| X)) = \overline{\text{create}}(d).\ E_\varphi(\varphi(d) \| X).$$

5. Concluding remarks

The message of this note is that process creation is a feature not too distant from process algebra. It should be stated that this introduction of process creation can equally well be applied within related formalisms like CCS [9], CSP [8], trace theory [10] provided sufficiently many recursion equations can be solved. In CCS it would be natural to write τ for $\overline{\text{create}}$(d).

References

[1] J.C.M. Baeten & J.A. Bergstra, Global renamings in concrete process algebra, report CS-R85.., Centre for Mathematics and Computer Science, Amsterdam 1985.

[2] J.C.M. Baeten, J.A. Bergstra & J.W. Klop, Syntax and defining equations for an interrupt mechanism in process algebra, Report CS-R8503, Centre for Mathematics and Computer Science, Amsterdam 1985.

[3] J.A. Bergstra, Put and get, primitives for synchronous unreliable message passing. Dept. of Philosophy, Univ. of Utrecht, report AL 85.., Utrecht 1985.

[4] J.A. Bergstra & J.W. Klop, Process algebra for synchronous communication, Information and Control 60 (1/3), pp. 109-137, 1984.

[5] J.A. Bergstra & J.W. Klop, Algebra of communicating processes with abstraction, TCS 37 (1985), 77-121, 1985.

[6] J.A. Bergstra & J.W. Klop, The algebra of recursively defined processes and the algebra of regular processes, in Proc. 11th Colloq. Automat. Lang. & Programming, Antwerpen (J. Paredaens, Ed.) Lect. Notes in Comp. Sc. No. 172, 82-94, Springer-Verlag, Berlin (1984).

[7] J.A. Bergstra, J.W. Klop & J.V. Tucker, Process algebra with asynchronous communication mechanisms, report CS-R8410, Centre for Mathematics and Computer Science, Amsterdam 1984.

[8] S.D. Brookes, C.A.R. Hoare & A.W. Roscoe, A theory of communicating sequential processes, J. Assoc. Comp. Mach. 31 (3), pp. 560-599, 1981.

[9] R. Milner, A calculus of communicating systems, Springer LNCS 92, 1980.

[10] M. Rem, Partially ordered computations with applications to VLSI design, in: Proc. 4th Advanced course on Found. of Comp. Sci. part 2, eds J.W. de Bakker & J. van Leeuwen, MC Tract 159, Mathematical Centre, pp. 1-44, Amsterdam 1983.