A Process Creation Mechanism in Process Algebra

J.A. Bergstra
Programming Research Group, University of Amsterdam
P.O.Box 41882, 1009 DB Amsterdam, The Netherlands
Department of Philosophy, State University of Utrecht
Heidelberglaan 2, 3584 CS Utrecht, The Netherlands

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. Extension of process algebra

In this paper we extend process algebra with a new operator that will be helpful to describe process creation. From a methodological point of view the extension of process algebra with new operators is just the right way to incorporate new features. Only in a very rich calculus with many operators one may hope to be able to perform significant algebraic calculations on systems. In many cases a new feature requires new (additional) syntax and more equations, only in very rare circumstances the addition of equations alone suffices to obtain an appropriate model of some new system aspect. 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. Process creation

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 $cr(d)$. The action $cr(d)$ stands for: create a process on basis of initial information d. Let $cr(D)$ denote the set $\{cr(d) | d \in D\}$.

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 $a|b = cr(d)$.

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

TABLE 1. Definition of E_φ

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 in terms of [1] we are dealing with concrete process algebra, i.e. there is no abstraction present.

EXAMPLE 2.1. $D = \{d\}$, $\varphi(d) = cr(d)$.
Let $P = E_\varphi(cr(d))$, then $P = \overline{cr}(d)\cdot E_\varphi(\varphi(d)) = \overline{cr}(d)\cdot E_\varphi(cr(d)) = \overline{cr}(d)\cdot P$. 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 AIP (see [1]):

$$\frac{\text{for all } n, \pi_n(X) = \pi_n(Y)}{X = Y}.$$

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

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

This leads to the second example.

EXAMPLE 2.2. Let $D = \{d\}$, $\varphi(d) = a\cdot cr(d) \| b\cdot cr(d)$, $a|b = \delta$ and $p = E_\varphi(cr(d))$. Then

$$p = \overline{cr}(d)\cdot E_\varphi(a\cdot cr(d) \| b\cdot cr(d)) \qquad \text{(using (*))}$$

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

EXAMPLE 2.3. Let $D=\{d\}$, $\varphi(d)=a\cdot(cr(d)\|cr(d))+b$, let $a|b=\delta$ and put $p=E_\varphi(cr(d))$. Now

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

(again using (*)).

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 (female) on D. Moreover there is a predicate V on D that indicates which genetic codes are vital and which are not. A vital genetic code will lead to living offspring whereas a non-vital code will not. Let for $a\in D$:

$$p^a = (\text{hunt}(a) + \text{sleep}(a) + \text{eat}(a) + \text{idle}(a))\cdot p^a + \text{end}(a).$$

Further for $a\in F$ we define the process q^a as follows:

$$q^a = \sum_{b\notin F}\text{pair}(a,b)\cdot(cr(a*b)\cdot q^a+\text{end}(a))+\text{end}(a).$$

On the other hand if $a\notin F$ then we define

$$q^a=\sum_{b\in F}\text{pair}(b,a)\cdot q^a+\text{end}(a).$$

Take the following communication function:

$$\text{end}(d)|\text{end}(d) = \overline{\text{end}}(d)$$
$$\text{pair}(a,b)|\text{pair}(a,b) = \overline{\text{pair}}(a,b)$$

(all other communications δ). Let

$$H_0 = \{\text{end}(d)|d\in D\}$$
$$H_1 = \{\text{pair}(a,b)|a,b\in D\}.$$

Then define

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

if a is vital (i.e. an element of V), otherwise take

$$\varphi(a) = \delta.$$

Now define the system S as follows:

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

S describes a population of animals 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 $B=\Sigma_{d\in D}\text{read}(d)\cdot cr(d)\cdot B$ and $\varphi(d) = \text{write}(d)$. Consider $B^* = E_\varphi(B)$ then

$$\begin{aligned}
B^* &= E_\varphi(\sum_{d\in D}\text{read}(d)\cdot cr(d)\cdot B)\\
&= \sum_{d\in D}\text{read}(d)\cdot\overline{cr}(d)\cdot E_\varphi(\varphi(d)\|B)\\
&= \sum_{d\in D}\text{read}(d)\cdot\overline{cr}(d)\cdot E_\varphi(\text{write}(d)\|B)\\
&= \sum_{d\in D}\text{read}(d)\cdot\overline{cr}(\text{write}(d)\cdot\delta\|E_\varphi(B))\\
&= \sum_{d\in D}\text{read}(d)\cdot\overline{cr}(d)\cdot(\text{write}(d)\|E_\varphi(B)\cdot\delta)\\
&= \sum_{d\in D}\text{read}\cdot\overline{cr}(d)\cdot(\text{write}(d)\|B^*)
\end{aligned}$$

Here we use the fact that B has no finite traces which implies that $E_\varphi(B)$ has no finite traces from which it follows that $E_\varphi(B)\cdot\delta$ equals $E_\varphi(B)$. Moreover we use the fact that in the absence of communication E_φ distributes over $\|$, as well as the identity $X\|(Y\cdot\delta)=(X\cdot\delta)\|Y$.

If we now use abstraction and substitute τ for $\overline{cr}(d)$ then we obtain (with $I=\{\overline{cr}(d)|d\in D\}$ and τ_I as in [5]) that $\tau_I(B^*)$ satisfies the equation for a bag over D:

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

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 Eratosthenes

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

3.3.1. Alphabet of actions A. δ: deadlock

t: internal step
for all $i,j \in \overline{N}$

write (i):	output i
send $i(j)$:	send j through port i
read $i(j)$:	read j through port i
cr(i):	create a new process from data i, we write $\overline{cr}(i)=t$

Then there is a family of atomic actions parametrized by pairs of elements of $\overline{N}$ as follows:

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

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

The communication function is taken as follows: 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 the SIEVE. We have SIEVE $=\partial_H E_\varphi(S_1)$ where H, φ and S_1 are given below.

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

$$S_1 = cr(2)\cdot \text{ send } 2(3) \ldots \text{send } 2(N)$$

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

$$\varphi(p)=S_p \text{with}$$

$$S_p = \text{write}(p).$$

$$\sum_{z \in \overline{N}} \text{read} p(z)\left[t(z=0 \bmod p)\cdot S_p + t(z \neq 0 \bmod p)\cdot cr(z)\cdot R_p^z\right]$$

$$R_p^i = \sum_{z \in \overline{N}} \text{read} p(z)\left[t(z=0 \bmod p)\cdot R_p^i + t(z \neq 0 \bmod p)\cdot \text{send } i(z)\cdot R_p^i\right]$$

The explanation of S_p is as follows: S_p will be created as soon as a new prime p is found by S_q (with q the prime preceding p). The first task of S_p is to output p, then it receives a sequence of larger numbers and checks all of these on being 0 mod p. The first $z \neq 0 \bmod p$ must be a prime since it has survived the entire pipeline from S_1 to S_2 to S_3 to S_5... till S_q. For this z a new process S_z is created. Thereafter S_p restricts itself to filtering out all numbers in the pipeline that are a multiple of p and transmitting the others to S_z.

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 $d \in D$ an action $cr^*(d)$ is introduced and communication works as follows:

$$cr(d) | cr^*(d) = \overline{cr}(d)$$

(the create-actions are not involved in any other proper communications). Now define K_φ as follows:

$$K_\varphi = \sum_{d \in D} cr^*(d) \cdot (K_\varphi \| \varphi(d)).$$

Let $H = \{cr(d), cr^*(d) | d \in D\}$. 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 any more. 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. We have to distinguish several cases

$\boxed{p = \delta}$ $\quad E_\varphi(p) = \delta$

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

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

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

Then there are the following cases for p involving the execution of an atomic action:

$$\left.\begin{array}{l} p = a \\ p = a\cdot X \end{array}\right\} a \text{ not of the form } cr(d)$$

$cr(d)$

$cr(d)\cdot X$

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

$\boxed{p = cr(d)\cdot X}$

$$E_\varphi(p) = \overline{cr}\ (d)\cdot E_\varphi(X \| \varphi(d))$$

$$\begin{aligned} \partial_H(K_\varphi \| p) &= \partial_H(\sum_{a\in D} cr^*(a)\cdot(K_\varphi \| \varphi(a)) \| cr(d)\cdot X) \\ &= \overline{cr}(d)\cdot\partial_H((K_\varphi \| \varphi(d)) \| X) \\ &= \overline{cr}(d)\cdot\partial_H(K_\varphi \| (\varphi(d) \| X)) = \overline{cr}(d)\cdot E_\varphi(\varphi(d) \| X) \end{aligned}$$

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], or trace theory [10] provided sufficiently many recursion equations can be solved. In CCS it would be natural to write τ for create (d).

References

1. J.C.M. Baeten, J.A. Bergstra (1988). Global renamings in concrete process algebra. *Information and Computation 78(3),* 205-245.
2. J.C.M. Baeten, J.A. Bergstra, J.W. Klop (1986). Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae IX (2),* 127-168.
3. J.A. Bergstra (1985). *Put and Get, Primitives for Synchronous Unreliable Message Passing,* Logic Group Preprint Series Nr. 3, CIF, State University of Utrecht.
4. J.A. Bergstra, J.W. Klop (1984). Process algebra for synchronous communication. *Information and Control 60 (1/3),* 109-137.
5. J.A. Bergstra, J.W. Klop (1985). Algebra of communicating processes with abstraction. *Theoretical Computer Science 37(1),* 77-121.
6. J.A. Bergstra, J.W. Klop (1984). The algebra of recursively defined processes and the algebra of regular processes. J. Paredaens (ed.). *Proc.*

11th Colloq. Automat. Lang. and Programming, Antwerpen, LNCS 172, Springer-Verlag, 82-94.

7. J.A. BERGSTRA, J.W. KLOP, J.V. TUCKER (1984). Process algebra with asynchronous communicating mechanisms. S.D. BROOKES, A.W. ROSCOE, G. WYNSKEL (eds.). *Proc. Seminar on Concurrency,* LNCS 197, Springer-Verlag, 76-95.
8. S.D. BROOKS, C.A.R. HOARE, A.W. ROSCOE (1981). A theory of communicating sequential processes. *J. Assoc. Comp. Mach. 31 (3),* 560-599.
9. R. MILNER (1980). *A Calculus of Communicating Systems,* LNCS 92, Springer-Verlag.
10. M. REM (1983). Partially ordered computations with applications to VLSI design. J.W. DE BAKKER, J. VAN LEEUWEN (eds.). *Proc. Found. of Comp. Sci. IV.2,* MC Tract 159, Centre for Mathematics and Computer Science, Amsterdam, 1-44.