Process Algebra Primitives for File Transfer*

Jan A. Bergstra[1,2] Alban Ponse[1]

[1] *University of Amsterdam, Programming Research Group*
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands

[2] *Utrecht University, Department of Philosophy*
P.O. Box 80126, 3508 TC Utrecht, The Netherlands

E-mail: `janb@wins.uva.nl` - `alban@wins.uva.nl`

Abstract

Action primitives for sequential file transfer are presented in the style of the early input action of [BB94].

1 Introduction

In modeling distributed systems one might associate infinite data types with 'local computation', and finite data types with information to be 'distributed'. For example, transferring a natural `N` from one location to another, say with help of an action `send(N)`, should not be regarded primitive in case `N` is a string of very large length over alphabet $\{0, 1, ..., 9\}$ (e.g., a representation of 9^{9^9} by juxtaposition of digits). For specification of (concurrent) transfer of files of arbitrary finite length, we propose new process algebra primitives. These are based on the finite alphabet from which 'files' are constructed, be it `ASCII`, $\{0, 1, ..., 9\}$, bits, or bytes.

In the next section we recall process algebra with early input. Then, in Section 3, we introduce our primitives for file transfer. In Section 4 we shortly discuss a notion of file manipulation primitives. The paper is ended with a short reflection on file transfer and manipulation in the realm of the ToolBus coordination architecture.

2 Early Input, Process Prefix, and SOS

In process algebra applications it is often useful to have specialised actions for various roles. With D a (finite) set of data items and I a collection of port names the following notations have become

*Dedicated to Paul Klint. His activities, professional both technical and managerial as well as personal have been appreciated very much by both authors, and have proven effective and useful for both of them. This paper appeared as as [BP97].

standard ([BK86]): for $d \in D$, $i \in I$

$$\begin{array}{ll} s_i(d): & \text{send } d \text{ along port } i, \\ r_i(d): & \text{receive } d \text{ along port } i, \\ c_i(d): & \text{communicate } d \text{ along port } i. \end{array}$$

The communication $s_i(d) \mid r_i(d) = c_i(d)$ describes the connection between these actions.

Let x, y, z be variables ranging over D. Process expressions with parameters in D are often denoted $P(x)$, $Q(x, y)$ etc. As an example, consider

$$P(x, y, z) = (s_i(x) \parallel s_j(y)) \cdot s_k(z)$$

and

$$Q(x, y) = r_i(x)^* s_j(d_0) \qquad (d_0 \in D, y \text{ need not occur in } Q(x, y)),$$

where the $\_^*\_$ operation is the binary Kleene star, defined by

$$P^*Q = P(P^*Q) + Q,$$

([Kle56], see also [BBP94]).

The expression $[e/x]P$ denotes the result of substituting e for x in a head normal form of P,

$$[e/x](\sum_{i \in K} a_i P_i + \sum_{j \in L} b_j) = \sum_{i \in K} [e/x]a_i \cdot [e/x]P_i + \sum_{j \in L} [e/x]b_j.$$

With $\sum_{d \in D} r_i(d) \cdot [d/x]P$ we obtain a process which reads a value $d \in D$ along port i and thereupon transforms itself into $[d/x]P$. This kind of process occurs very frequently indeed, whence [BB94] proposed the notation

$$er_i^D(x); P = \sum_{d \in D} r_i(d) \cdot [d/x]P.$$

Here er_i^D is an *early read* action, and $\_;\_$ is the *action prefix operation.* Of course, $[e/x]er_i(x); P = er_i(x); P$, as x is in this case not free for substitution.

Action prefix is extended to *process prefix* satisfying for action set A:

$$\begin{array}{rcll} (x + y); z & = & x; z + y; z & \\ (a \cdot x); y & = & a; (x; y) & (a \in A) \\ a; x & = & a \cdot x & (a \in A \setminus \{\text{early read actions}\}). \end{array}$$

A nice bonus of the early input notation is that parallel input is easily described.

$$
\begin{aligned}
&\left(er_1^D(x) \parallel er_2^D(y)\right) ; P(x,y)\\
&\quad = \left(er_1^D(x) \cdot er_2^D(y) \ + er_2^D(y) \cdot er_1^D(x) + \delta\right) ; P(x,y)\\
&\quad = \left(er_1^D(x) \cdot er_2^D(y)\right) ; P(x,y) \ +\\
&\quad\ \left(er_2^D(y) \cdot er_1^D(x)\right) ; P(x,y)\\
&\quad = er_1^D(x); (er_2^D(y); P(x,y)) \ +\\
&\quad\ er_2^D(y); (er_1^D(x); P(x,y))\\
&\quad = \textstyle\sum_{d\in D} r_1(d) \cdot \sum_{e\in D} \cdot r_2(e) \cdot P(d,e) \ +\\
&\quad\ \textstyle\sum_{e\in D} r_2(e) \cdot \sum_{d\in D} \cdot r_1(d) \cdot P(d,e).
\end{aligned}
$$

Indeed parallel input can be expressed without early input actions, but at cost of a combinatorial explosion. Expressions like

$$\left(er_1^D(x_1) \parallel ... \parallel er_n^D(x_n)\right) ; P(x_1, ..., x_n)$$

are instrumental for the description of data flow networks and grid protocols (see [BHP97], inspired by [TT94]). Avoiding early input actions at cost of expanding the merge is hardly a practical option.

Finally, we display in Table 1 transition rules that can be used to assign a transition system to process expressions. In this table, P and Q, and their primed versions range over process expressions with parameters in data set D.

3 Primitives for File Transfer

We now turn attention to larger pieces of data which cannot be transferred in one step. We consider D^*, writing Λ for the empty sequence, $\_ +\!\!\!+ \_$ for strong concatenation, $\_\frown\_$ for left insertion of a D-element in a sequence, and $\_\smile\_$ for append (right insertion). We use X as a variable ranging over D^*.

Sending or receiving a file $\sigma \in D^*$ will require one action per item in σ. For *file output* we propose the following notation:

ofs_i^D : open file send along port i, an action that initiates sending a file over D along port i,

$\mathit{Sf}_i^D(\sigma)$: a process that represents the actual sending of the elements of file σ over D, specified by:

$$
\begin{aligned}
\mathit{Sf}_i^D(\Lambda) &= \mathit{cfs}_i^D,\\
\mathit{Sf}_i^D(d \frown \sigma) &= s_i(d) \cdot \mathit{Sf}_i^D(\sigma),
\end{aligned}
$$

cfs_i^D : close file send along port i, an action indicating that end of file (`EOF`) has been reached.

Table 1: Transition rules for process algebra with early input.

	$a \xrightarrow{a} \surd \quad (a \in A \setminus \{\text{early read actions}\})$	
$er_i^D(x);P$	$er_i^D(x);P \xrightarrow{r_i(d)} [d/x]P \quad (d \in D)$	
$+, \cdot, \mathbin{\Vert\!\lfloor}, \Vert, {}^*$	$\dfrac{P \xrightarrow{a} \surd}{\begin{array}{l} P+Q \xrightarrow{a} \surd \\ Q+P \xrightarrow{a} \surd \\ P \cdot Q \xrightarrow{a} Q \\ P \mathbin{\Vert\!\lfloor} Q \xrightarrow{a} Q \\ P \Vert Q \xrightarrow{a} Q \\ Q \Vert P \xrightarrow{a} Q \\ P^*Q \xrightarrow{a} P^*Q \\ Q^*P \xrightarrow{a} \surd \end{array}}$	$\dfrac{P \xrightarrow{a} P'}{\begin{array}{l} P+Q \xrightarrow{a} P' \\ Q+P \xrightarrow{a} P' \\ P \cdot Q \xrightarrow{a} P'Q \\ P \mathbin{\Vert\!\lfloor} Q \xrightarrow{a} P' \Vert Q \\ P \Vert Q \xrightarrow{a} P' \Vert Q \\ Q \Vert P \xrightarrow{a} Q \Vert P' \\ P^*Q \xrightarrow{a} P'(P^*Q) \\ Q^*P \xrightarrow{a} P' \end{array}}$
$\mid, \Vert$ $a \mid b = c \in A$	$\dfrac{P \xrightarrow{a} \surd \quad Q \xrightarrow{b} \surd}{\begin{array}{c} P \mid Q \xrightarrow{c} \surd \\ P \Vert Q \xrightarrow{c} \surd \end{array}}$	$\dfrac{P \xrightarrow{a} \surd \quad Q \xrightarrow{b} Q'}{\begin{array}{c} P \mid Q \xrightarrow{c} Q' \\ P \Vert Q \xrightarrow{c} Q' \end{array}}$
	$\dfrac{P \xrightarrow{a} P' \quad Q \xrightarrow{b} \surd}{\begin{array}{c} P \mid Q \xrightarrow{c} P' \\ P \Vert Q \xrightarrow{c} P' \end{array}}$	$\dfrac{P \xrightarrow{a} P' \quad Q \xrightarrow{b} Q'}{\begin{array}{c} P \mid Q \xrightarrow{c} P' \Vert Q' \\ P \Vert Q \xrightarrow{c} P' \Vert Q' \end{array}}$
∂_H	$\dfrac{P \xrightarrow{a} \surd}{\partial_H(P) \xrightarrow{a} \surd}$ if $a \notin H$	$\dfrac{P \xrightarrow{a} P'}{\partial_H(P) \xrightarrow{a} \partial_H(P')}$ if $a \notin H$

For *file input* we use

ofr_i^D : open file receive along port i, an action that marks the start of a file transmission,

cfr_i^D : close file receive along port i, an action indicating that end of file (`EOF`) has been reached.

Then, similar to the early input case we propose the following primitives for *early file input*:

$$\begin{array}{rcl} ofr_i^D(X); P & = & ofr_i^D \cdot [\Lambda/X]P, \text{ i.e., bind } X \text{ in } P \text{ for file input,} \\ efr_i^D(X); P & = & \sum_{d\in D} r_i(d) \cdot [X^\frown d/X]P, \\ & & \text{i.e., append in } P \text{ the next item to } X, \\ cfr_i^D(X); P & = & cfr_i^D \cdot P. \end{array}$$

Here the substitution $[X^\frown d/X]$ must be handled with care: it does not affect $efr_i^D(X)$ but substitutes $X^\frown d$ for X in all other occurrences. So, unlike the case of the early input $er_i^D(x); P$ where x can be considered bound, X is not bound in $efr_i^D(X); P$.

As a useful macro we have

$$Efr_i^D(X) \stackrel{\text{def}}{=} efr_i^D(X)^* cfr_i.$$

Writing $\overline{P(X)} = Efr_i^D(X); P(X)$ we can derive

$$\begin{array}{rcl} \overline{P(X)} & = & cfr_i^D \cdot P(X) + \sum_{d\in D} r_i(d) \cdot (\\ & & \quad cfr_i^D \cdot [X^\frown d/X]P + \sum_{e\in D} r_i(e) \cdot (\\ & & \quad\quad cfr_i^D \cdot [X^\frown d^\frown e/X]P) \ + ... \\ & &)). \end{array}$$

As an example, consider the following derivation:

$$\begin{array}{l} [X^\frown d_0/X]\overline{P(X)} \\ \quad = \ [X^\frown d_0/X] \left(cfr_i^D \cdot P(X) + \sum_{d\in D} r_i(d) \cdot [X^\frown d/X]\overline{P(X)} \right) \\ \quad = \ cfr_i^D \cdot P(X^\frown d_0) + \sum_{d\in D} r_i(d) \cdot [X^\frown d_0/X][X^\frown d/X]\overline{P(X)} \\ \quad = \ cfr_i^D \cdot P(X^\frown d_0) + \sum_{d\in D} r_i(d) \cdot [X^\frown d_0{}^\frown d/X]\overline{P(X)}. \end{array}$$

Now $ofr_i^D(X)$ acts as a binder, and in the most useful combination

$$\left(ofr_i^D(X) \cdot Efr_i^D(X) \right); P(X)$$

X is bound indeed. Let H be such that communications along channel i are enforced:

$$H \ = \ \{ofs_i^D, ofr_i^D, cfs_i^D, cfr_i^D\} \ \cup \ \{s_i(d), r_i(d) \mid d \in D\}.$$

Let furthermore $\sigma = d_1{}^\frown d_2 ...{}^\frown d_n$, then we obtain

$$\begin{array}{l} \partial_H(ofs_i^D \cdot Sf_i(\sigma) \parallel (ofr_i^D(X) \cdot Efr_i^D(X)); P(X)) \\ \qquad = \ ofc_i^D \cdot \partial_H(Sf_i(\sigma) \parallel [\Lambda/X]\overline{P(X)}) \\ \qquad = \ ofc_i^D \cdot c_i(d_1) \cdot ... \cdot c_i(d_n) \cdot cfc_i^D \cdot P(d_1{}^\frown d_2 ...{}^\frown d_n). \end{array}$$

For the specific case $\sigma = d_1 \frown d_2 \in D^*$ and $P(X) = \mathit{ofs}_j^D \cdot \mathit{Sf}_j^D(X^R)$, we derive this identity in full detail in Table 2. Here σ^R is the *reversal* of σ, defined by $\Lambda^R = \Lambda$, $(\sigma \smile d)^R = d \frown \sigma^R$.

Table 2: Derivation of a file transfer.

$$
\begin{aligned}
&\partial_H(\mathit{ofs}_i^D \cdot \mathit{Sf}_i(d_1 \frown d_2) \parallel (\mathit{ofr}_i^D(X) \cdot \mathit{Efr}_i^D(X); [\mathit{ofs}_j^D \cdot \mathit{Sf}_j^D(X^R)])\\
&= \quad \mathit{ofc}_i^D \cdot \partial_H(\mathit{Sf}_i(d_1 \frown d_2) \parallel [\Lambda/X]\mathit{Efr}_i^D(X); [\mathit{ofs}_j^D \cdot \mathit{Sf}_j^D(X^R)])\\
&= \quad \mathit{ofc}_i^D \cdot\\
&\quad \partial_H\left(\mathit{Sf}_i(d_1 \frown d_2) \parallel \left[\begin{array}{l} \mathit{cfr}_i^D \cdot \mathit{ofs}_j^D \cdot \mathit{Sf}_j^D(\Lambda)\\ +\\ \sum_{d\in D} r_i(d) \cdot [d/X]\mathit{Efr}_i^D(X); [\mathit{ofs}_j^D \cdot \mathit{Sf}_j^D(X^R)] \end{array}\right]\right)\\
&= \quad \mathit{ofc}_i^D \cdot c_i(d_1) \cdot \partial_H(\mathit{Sf}_i(d_2) \parallel [d_1/X]\mathit{Efr}_i^D(X); [\mathit{ofs}_j^D \cdot \mathit{Sf}_j^D(X^R)])\\
&= \quad \mathit{ofc}_i^D \cdot c_i(d_1) \cdot\\
&\quad \partial_H\left(\mathit{Sf}_i(d_2) \parallel \left[\begin{array}{l} \mathit{cfr}_i^D \cdot \mathit{ofs}_j^D \cdot \mathit{Sf}_j^D(d_1)\\ +\\ \sum_{d\in D} r_i(d) \cdot [d_1 \smile d/X]\mathit{Efr}_i^D(X); [\mathit{ofs}_j^D \cdot \mathit{Sf}_j^D(X^R)] \end{array}\right]\right)\\
&= \quad \mathit{ofc}_i^D \cdot c_i(d_1) \cdot c_i(d_2) \cdot \partial_H(\mathit{cfs}_i^D \parallel [d_1 \smile d_2/X]\mathit{Efr}_i^D(X); [\mathit{ofs}_j^D \cdot \mathit{Sf}_j^D(X^R)])\\
&= \quad \mathit{ofc}_i^D \cdot c_i(d_1) \cdot c_i(d_2) \cdot\\
&\quad \partial_H\left(\mathit{cfs}_i^D \parallel \left[\begin{array}{l} \mathit{cfr}_i^D \cdot \mathit{ofs}_j^D \cdot \mathit{Sf}_j^D(d_2 \frown d_1)\\ +\\ \sum_{d\in D} r_i(d) \cdot [d_1 \smile d_2 \smile d/X]\mathit{Efr}_i^D(X); [\mathit{ofs}_j^D \cdot \mathit{Sf}_j^D(X^R)] \end{array}\right]\right)\\
&= \quad \mathit{ofc}_i^D \cdot c_i(d_1) \cdot c_i(d_2) \cdot \mathit{cfc}_i^D \cdot \mathit{ofs}_j^D \cdot \mathit{Sf}_j^D(d_2 \frown d_1).
\end{aligned}
$$

For these file transfer primitives we give the transition rules displayed in Table 3. For $\mathit{Efr}_i^D(X); P$ we can derive the following transitions:

$$
\begin{aligned}
&\mathit{Efr}_i^D(X); P \xrightarrow{\mathit{ofr}_i^D} [\Lambda/X]P\\
&\mathit{Efr}_i^D(X); P \xrightarrow{r_i(d)} [X \smile d/X]\mathit{Efr}_i^D(X); P \qquad (d \in D).
\end{aligned}
$$

4 File Manipulation Primitives

File transfer primitives will naturally lead to file manipulation primitives.

Let $F : V \to W_{\mathtt{D}}$ ($W_{\mathtt{D}} = W \cup \{\mathtt{D}\}$), with $F(v) = \mathtt{D}$ representing the case that the computation $F(v)$

Table 3: Transition rules for process algebra with early file input.

$ofr_i^D(X); P$	$ofr_i^D(X); P \xrightarrow{ofr_i^D} [\Lambda/X]P$
$efr_i^D(x); P$	$efr_i^D(X); P \xrightarrow{r_i(d)} [X^\frown d/X]P \qquad (d \in D)$

diverges. The special constant $\mathtt{D}$ is taken from [BBR95]. We write:

$$\begin{aligned} Y_{F,j}(v) &= \iota^2 \cdot s_j(w) \quad &\text{if} \quad F(v) = w \in W, \\ Y_{F,j}(v) &= \iota^\omega &\text{if} \quad F(v) = \mathtt{D}. \end{aligned}$$

Here Y stands for "yield", and ι is an action that represents internal activity (ι satisfies the law $x \cdot \iota = x \cdot \iota \cdot \iota$).

Now $er_k(X); Y_{F,j}(X)$ represents the function F as a process. For X of sort D^* we use $Efr_k^D(X); Y_{F,j}(X)$.

For a nondeterministic operation (relation) R it is not so easy to find a general form for representing it as a process. We have to deal with a computation process that may diverge as well as deliver different values (along different paths). In special cases however something can be said. We will discuss an example involving the arbitrary (random) permutation of lists in advance of the computation of output. For instance the action prefix $Aperm^D(X)$ can be defined as follows:

$$Aperm^D(X); P = \sum_{r:perm(X)} \iota^2 \cdot [r(X)/X]P.$$

Here $perm(X)$ is the set of permutations of the list X. Clearly $|perm(X)| = length(X)!$.

If F computes a value in W from a sequence X of D's, then

$$\left(ofr_i^D(X) \cdot Efr_i^D(X) \cdot Aperm^D(X) \cdot \iota \right); Y_{F,j}(X)$$

is a process that opens file X, and reads its values until end of file along port i. Thereafter it generates an arbitrary permutation, evaluates function F on that permutation, and outputs the result along port j.

Of course this operation has many variations, e.g., producing even permutations ($Aperm_{even}^D(X)$), or cyclic permutations ($Aperm_{cyclic}^D(X)$) only.

5 Digression

In case it is known that a file to be received has a bound on its length, we can use much more specific operations, both for modeling its representation, and for modeling its transfer. We define *bounded* *repetition* $x^n y$ by

$$\begin{aligned} x^0 y &= y, \\ x^{n+1} y &= x \cdot (x^n y), \end{aligned}$$

and *bounded iteration* (or bounded binary Kleene star) $x^{*^n} y$ by

$$\begin{aligned} x^{*^0} y &= y, \\ x^{*^{n+1}} y &= x \cdot (x^{*^n} y) + y. \end{aligned}$$

For a file that has exactly length n one may use the prefix

$$\left(efr_i^D (X)^n \, cfr_i^D \right) ; P.$$

In the case that one knows that a file to be received has at most n elements, the prefix

$$\left(efr_i^D (X)^{*^n} cfr_i^D \right) ; P$$

will work. Though such modeling will be closer to reality, it may still be worthwhile to consider $\_^*\_$ and $\_^*\delta$ as the 'limit' cases of $x^{*^n} y$ and $x^n y$, respectively. Proving a certain property defined in terms of these more abstract operations might be easier. Still, such a property might give one sufficient information about properties that are expressed with the specific operations discussed above.

Giving a realistic example of the use of our file transfer primitives is not that easy. Clearly in Paul Klint's favorite subject (distributed programming environments, [Kli93]), files of "arbitrary" length play a key role as declarations, programs, variable environments, typechecker outputs etc. are all character strings, say from type `C`. A component like

$$\begin{aligned} &\left(ofr_1^{\mathtt{C}} \cdot Efr_1^{\mathtt{C}}(\mathtt{decls}) \parallel ofr_2^{\mathtt{C}} \cdot Efr_2^{\mathtt{C}}(\mathtt{stats}) \right) ; \\ &\quad \left(ofs_3^{\mathtt{C}} \cdot Sf_3^{\mathtt{C}}(\mathtt{decls} +\!\!\!+ \mathtt{stats}) \parallel ofs_4^{\mathtt{C}} \cdot Sf_4^{\mathtt{C}}(\mathtt{type\text{-}check}(\mathtt{decls} +\!\!\!+ \mathtt{stats})) \right) \end{aligned}$$

can input in parallel strings like `decls` (declarations) and `stats` (statements), and return an encoded parse tree and a file with type check output.

The ToolBus model using process algebra as given in [BK96a, BK96b] is an approximation of reality rather than a dogma. It so seems that this model would get closer to the facts if the communication between bus and tools were described by means of the file transfer primitives established above, rather than the atomic action used in [BK96a].

References

[BB94] J.C.M. Baeten and J.A. Bergstra. On sequential composition, action prefixes and process prefix. *Journal of Formal Aspects of Computing Science*, 6(3):83–98, 1994.

[BBP94] J.A. Bergstra, I. Bethke, and A. Ponse. Process algebra with iteration and nesting. *Computer Journal*, 37(4):243–258, 1994.

[BBR95] J.A. Bergstra, I. Bethke, and P.H. Rodenburg. A propositional logic with 4 values: true, false, divergent and meaningless. *Journal of Applied Non-Classical Logics*, 5:199–217, 1995.

[BHP97] J.A. Bergstra, J.A. Hillebrand, and A. Ponse. Grid protocols based on synchronous communication, *Science of Computer Programming*, 29:199-233, 1997.

[BK86] J.A. Bergstra and J.W. Klop. Process algebra: specification and verification in bisimulation semantics. In M. Hazewinkel, J.K. Lenstra, and L.G.L.T. Meertens, editors, *Mathematics and Computer Science II*, CWI Monograph 4, pages 61–94. North-Holland, Amsterdam, 1986.

[BK96a] J.A. Bergstra and P. Klint. The ToolBus coordination architecture. In P. Ciancarini and C. Hankin, editors, *Coordination Languages and Models (COORDINATION), LNCS 1061*, pages 75–88. Springer-Verlag, 1996.

[BK96b] J.A. Bergstra and P. Klint. The discrete time ToolBus. In M. Wirsing and M. Nivat, editors, *Algebraic Methodology and Software Technology (AMAST), LNCS 1101*, pages 286–305. Springer-Verlag, 1996.

[BP97] J.A. Bergstra and A. Ponse. Process algebra primitives for file transfer. Liber Amicorum dedicated to Paul Klint, pages 33–42, CWI, 1997.

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.

[Kle56] S. C. Kleene. Representation of events in nerve nets and finite automata. In *Automata Studies*, pages 3–41. Princeton University Press, Princeton NJ, 1956.

[Kli93] P. Klint. A meta-environment for generating programming environments. ACM Transactions on Software Engineering and Methodology, 2(2):176–201, 1993.

[TT94] B.C. Thompson and J.V. Tucker. Equational specification of Synchronous Concurrent Algorithms and architectures (Second Edition). Report CSR 15-94, University of Wales, Swansea, 1994.