University of Amsterdam

Programming Research Group

Transmission Protocols for Instruction Streams

J.A. Bergstra
C.A. Middelburg

Report PRG0903 February 2009

J.A. Bergstra

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ   Amsterdam
The Netherlands

tel. +31 20 525.7591
e-mail: janb@science.uva.nl

C.A. Middelburg

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ   Amsterdam
The Netherlands

e-mail: kmiddelb@science.uva.nl

Programming Research Group Electronic Report Series

Transmission Protocols for Instruction Streams

J.A. Bergstra and C.A. Middelburg

Informatics Institute, Faculty of Science, University of Amsterdam,
Science Park 107, 1098 XG Amsterdam, the Netherlands
J.A.Bergstra@uva.nl,C.A.Middelburg@uva.nl

Abstract. Threads as considered in thread algebra model behaviours to be controlled by some execution environment: upon each action performed by a thread, a reply from its execution environment – which takes the action as an instruction to be processed – determines how the thread proceeds. In this paper, we are concerned with the case where the execution environment is remote: we describe and analyse some transmission protocols for passing instructions from a thread to a remote execution environment.

Keywords: transmission protocol, instruction stream, thread algebra, process algebra, process extraction.

1998 ACM Computing Classification: D.2.1, D.2.4, F.1.1, F.3.1.

1 Introduction

The behaviours produced by sequential programs under execution are behaviours to be controlled by some execution environment. The execution environment concerned is increasingly more a remote execution environment. The objective of the current paper is to clarify the phenomenon of remotely controlled program behaviours.

Basic thread algebra [7], BTA in short, is a form of process algebra tailored to the description and analysis of the behaviours produced by sequential programs under execution.[1] Threads as considered in basic thread algebra model behaviours to be controlled by some execution environment. Threads proceed by performing steps, called basic actions in what follows, in a sequential fashion. The execution environment of a thread takes the basic actions performed by the thread as instructions to be processed. Upon each basic action performed by the thread, a reply from the execution environment determines how the thread proceeds. To achieve the objective of the current paper, we study some transmission protocols for passing instructions from a thread to a remote execution environment.

General process algebras, such as ACP [6, 4], CCS [11, 13] and CSP [9, 12], are too general for the description and analysis of the behaviours produced by

[1] In [7], basic thread algebra is introduced under the name basic polarized process algebra.

sequential programs under execution. That is, it is quite awkward to describe and analyse behaviours of this kind using such a general process algebra. However, the behaviours considered in basic thread algebra can be viewed as processes that are definable over ACP, see e.g. [8]. This allows for the transmission protocols mentioned above to be described and their correctness to be verified using ACP or rather ACP$^\tau$, an extension of ACP which supports abstraction from internal actions. We consider first a very simple transmission protocol and then a more complex one that is more efficient.

This paper is organized as follows. First, we give brief summaries of BTA (Section 2) and ACP$^\tau$ (Section 3). Next, we make mathematically precise the connection between behaviours as considered in BTA and processes as considered in ACP$^\tau$ (Section 4). After that, we describe and analyse the above-mentioned transmission protocols (Sections 5 and 6). Finally, we make some concluding remarks (Section 7).

2 Thread Algebra

In this section, we review BTA (Basic Thread Algebra). BTA is concerned with behaviours as exhibited by sequential programs under execution. These behaviours are called threads.

In BTA, it is assumed that a fixed but arbitrary set $\mathcal{A}$ of *basic actions* has been given. A thread performs basic actions in a sequential fashion. Upon each basic action performed, a reply from the execution environment of the thread determines how it proceeds. The possible replies are the Boolean values T and F.

To build terms, BTA has the following constants and operators:

– the *deadlock* constant D;
– the *termination* constant S;
– for each $a \in \mathcal{A}$, the binary *postconditional composition* operator $\unlhd a \unrhd$.

We assume that there are infinitely many variables, including x, y, z. Terms are built as usual. We use infix notation for the postconditional composition operator. We introduce *basic action prefixing* as an abbreviation: $a \circ p$, where $a \in \mathcal{A}$ and p is a BTA term, abbreviates $p \unlhd a \unrhd p$.

The thread denoted by a closed term of the form $p \unlhd a \unrhd q$ will first perform a, and then proceed as the thread denoted by p if the reply from the execution environment is T and proceed as the thread denoted by q if the reply from the execution environment is F. The threads denoted by D and S will become inactive and terminate, respectively. This implies that each closed BTA term denotes a thread that will become inactive or terminate after it has performed finitely many basic actions. Infinite threads can be described by guarded recursion.

A *guarded recursive specification* over BTA is a set of recursion equations $E = \{X = t_X \mid X \in V\}$, where V is a set of variables and each t_X is a BTA term of the form D, S or $t \unlhd a \unrhd t'$ with t and t' that contain only variables from V. We write $\mathrm{V}(E)$ for the set of all variables that occur in E. We are

Table 1. Axioms for guarded recursion

$\langle X \mid E\rangle = \langle t_X \mid E\rangle$	if $X = t_X \in E$	RDP
$E \Rightarrow X = \langle X \mid E\rangle$	if $X \in \mathrm{V}(E)$	RSP

only interested in models of BTA in which guarded recursive specifications have unique solutions, such as the projective limit model of BTA presented in [5].

For each guarded recursive specification E and each $X \in \mathrm{V}(E)$, we introduce a constant $\langle X \mid E\rangle$ standing for the unique solution of E for X. The axioms for these constants are given in Table 1. In this table, we write $\langle t_X \mid E\rangle$ for t_X with, for all $Y \in \mathrm{V}(E)$, all occurrences of Y in t_X replaced by $\langle Y \mid E\rangle$. X, t_X and E stand for an arbitrary variable, an arbitrary BTA term and an arbitrary guarded recursive specification over BTA, respectively. Side conditions are added to restrict what X, t_X and E stand for.

In the sequel, we will make use of a version of BTA in which the following additional assumptions relating to $\mathcal{A}$ are made: (i) a fixed but arbitrary set $\mathcal{F}$ of *foci* has been given; (ii) a fixed but arbitrary set $\mathcal{M}$ of *methods* has been given; (iii) $\mathcal{A} = \{f.m \mid f \in \mathcal{F}, m \in \mathcal{M}\}$. These assumptions are based on the view that the execution environment provides a number of services. Performing a basic action $f.m$ is taken as making a request to the service named f to process command m. As usual, we will write $\mathbb{B}$ for the set $\{\mathsf{T}, \mathsf{F}\}$.

3 Process Algebra

In this section, we review ACP$^\tau$ (Algebra of Communicating Processes with abstraction). This is the process algebra that will be used in Section 4 to make precise what processes are produced by the threads denoted by closed terms of BTA with guarded recursion. For a comprehensive overview of ACP$^\tau$, the reader is referred to [4, 10].

In ACP$^\tau$, it is assumed that a fixed but arbitrary set A of *atomic actions*, with $\tau, \delta \notin \mathsf{A}$, and a fixed but arbitrary commutative and associative function $\mid : \mathsf{A} \times \mathsf{A} \to \mathsf{A} \cup \{\delta\}$ have been given. The function $\mid$ is regarded to give the result of synchronously performing any two atomic actions for which this is possible, and to give δ otherwise. In ACP$^\tau$, τ is a special atomic action, called the silent step. The act of performing the silent step is considered unobservable. Because it would otherwise be observable, the silent step is considered an atomic action that cannot be performed synchronously with other atomic actions.

ACP$^\tau$ has the following constants and operators:

- for each $e \in \mathsf{A}$, the *atomic action* constant e ;
- the *silent step* constant τ ;
- the *deadlock* constant δ ;
- the binary *alternative composition* operator $+$;
- the binary *sequential composition* operator $\cdot$;
- the binary *parallel composition* operator $\parallel$;

– the binary *left merge* operator $\mathbin{\|\!\!\lfloor}$;
– the binary *communication merge* operator $\mid$;
– for each $H \subseteq \mathsf{A}$, the unary *encapsulation* operator ∂_H ;
– for each $I \subseteq \mathsf{A}$, the unary *abstraction* operator τ_I .

We assume that there are infinitely many variables, including x, y, z. Terms are built as usual. We use infix notation for the binary operators.

Let p and q be closed ACP$^\tau$ terms, $e \in \mathsf{A}$, and $H, I \subseteq \mathsf{A}$. Intuitively, the constants and operators to build ACP$^\tau$ terms can be explained as follows:

– e first performs atomic action e and next terminates successfully;
– τ performs an unobservable atomic action and next terminates successfully;
– δ can neither perform an atomic action nor terminate successfully;
– $p + q$ behaves either as p or as q, but not both;
– $p \cdot q$ first behaves as p and on successful termination of p it next behaves as q;
– $p \parallel q$ behaves as the process that proceeds with p and q in parallel;
– $p \mathbin{\|\!\!\lfloor} q$ behaves the same as $p \parallel q$, except that it starts with performing an atomic action of p;
– $p \mid q$ behaves the same as $p \parallel q$, except that it starts with performing an atomic action of p and an atomic action of q synchronously;
– $\partial_H(p)$ behaves the same as p, except that atomic actions from H are blocked;
– $\tau_I(p)$ behaves the same as p, except that atomic actions from I are turned into unobservable atomic actions.

The axioms of ACP$^\tau$ are given in Table 2. CM2–CM3, CM5–CM7, C1–C4, D1–D4 and TI1–TI4 are actually axiom schemas in which a, b and c stand for arbitrary constants of ACP$^\tau$, and H and I stand for arbitrary subsets of A.

A *recursive specification* over ACP$^\tau$ is a set of recursion equations $E = \{X = t_X \mid X \in V\}$, where V is a set of variables and each t_X is an ACP$^\tau$ term containing only variables from V. Let t be an ACP$^\tau$ term without occurrences of abstraction operators containing a variable X. Then an occurrence of X in t is *guarded* if t has a subterm of the form $e \cdot t'$ where $e \in \mathsf{A}$ and t' is a term containing this occurrence of X. Let E be a recursive specification over ACP$^\tau$. Then E is a *guarded recursive specification* if, in each equation $X = t_X \in E$: (i) abstraction operators do not occur in t_X and (ii) all occurrences of variables in t_X are guarded or t_X can be rewritten to such a term using the axioms of ACP$^\tau$ in either direction and/or the equations in E except the equation $X = t_X$ from left to right. We only consider models of ACP$^\tau$ in which guarded recursive specifications have unique solutions, such as the models of ACP$^\tau$ presented in [4].

For each guarded recursive specification E and each variable X that occurs in E, we introduce a constant $\langle X|E\rangle$ standing for the unique solution of E for X. The axioms for these constants are RDP and RSP given in Table 3. In RDP, we write $\langle t_X|E\rangle$ for t_X with, for all $Y \in \mathrm{V}(E)$, all occurrences of Y in t_X replaced by $\langle Y|E\rangle$. RDP and RSP are actually axiom schemas in which X stands for an

Table 2. Axioms of ACP$^\tau$

$x + y = y + x$	A1	$x \cdot \tau = x$		B1
$(x + y) + z = x + (y + z)$	A2	$x \cdot (\tau \cdot (y + z) + y) = x \cdot (y + z)$		B2
$x + x = x$	A3			
$(x + y) \cdot z = x \cdot z + y \cdot z$	A4	$\partial_H(a) = a$	if $a \notin H$	D1
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	A5	$\partial_H(a) = \delta$	if $a \in H$	D2
$x + \delta = x$	A6	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$		D3
$\delta \cdot x = \delta$	A7	$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$		D4
$x \parallel y = x \mathbin{⌊\!⌊} y + y \mathbin{⌊\!⌊} x + x \mid y$	CM1	$\tau_I(a) = a$	if $a \notin I$	TI1
$a \mathbin{⌊\!⌊} x = a \cdot x$	CM2	$\tau_I(a) = \tau$	if $a \in I$	TI2
$a \cdot x \mathbin{⌊\!⌊} y = a \cdot (x \parallel y)$	CM3	$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$		TI3
$(x + y) \mathbin{⌊\!⌊} z = x \mathbin{⌊\!⌊} z + y \mathbin{⌊\!⌊} z$	CM4	$\tau_I(x \cdot y) = \tau_I(x) \cdot \tau_I(y)$		TI4
$a \cdot x \mid b = (a \mid b) \cdot x$	CM5			
$a \mid b \cdot x = (a \mid b) \cdot x$	CM6	$a \mid b = b \mid a$		C1
$a \cdot x \mid b \cdot y = (a \mid b) \cdot (x \parallel y)$	CM7	$(a \mid b) \mid c = a \mid (b \mid c)$		C2
$(x + y) \mid z = x \mid z + y \mid z$	CM8	$\delta \mid a = \delta$		C3
$x \mid (y + z) = x \mid y + x \mid z$	CM9	$\tau \mid a = \delta$		C4

Table 3. RDP, RSP and AIP

$\langle X \mid E \rangle = \langle t_X \mid E \rangle$	if $X = t_X \in E$	RDP	$\pi_0(a) = \delta$	PR1
$E \Rightarrow X = \langle X \mid E \rangle$	if $X \in \mathrm{V}(E)$	RSP	$\pi_{n+1}(a) = a$	PR2
			$\pi_0(a \cdot x) = \delta$	PR3
$\bigwedge_{n \geq 0} \pi_n(x) = \pi_n(y) \Rightarrow x = y$		AIP	$\pi_{n+1}(a \cdot x) = a \cdot \pi_n(x)$	PR4
			$\pi_n(x + y) = \pi_n(x) + \pi_n(y)$	PR5
			$\pi_n(\tau) = \tau$	PR6
			$\pi_n(\tau \cdot x) = \tau \cdot \pi_n(x)$	PR7

arbitrary variable, t_X stands for an arbitrary ACP$^\tau$ term, and E stands for an arbitrary guarded recursive specification over ACP$^\tau$.

Closed terms of ACP with guarded recursion that denote the same process cannot always be proved equal by means of the axioms of ACP together with RDP and RSP. To remedy this, we introduce AIP (Approximation Induction Principle). AIP is based on the view that two processes are identical if their approximations up to any finite depth are identical. The approximation up to depth n of a process behaves the same as that process, except that it cannot perform any further atomic action after n atomic actions have been performed. AIP is given in Table 3. Here, approximation up to depth n is phrased in terms of a unary *projection* operator π_n. The axioms for these operators are axioms PR1–PR7 in Table 3. PR1–PR7 are actually axiom schemas in which a stands

Table 4. Defining equations for process extraction operation

$\lvert X\rvert^{\mathrm{c}} = X$
$\lvert \mathsf{S}\rvert^{\mathrm{c}} = \mathrm{stop}$
$\lvert \mathsf{D}\rvert^{\mathrm{c}} = \mathrm{i} \cdot \delta$
$\lvert t_1 \trianglelefteq f.m \trianglerighteq t_2\rvert^{\mathrm{c}} = \mathrm{s}_f(m) \cdot (\mathrm{r}_f(\mathsf{T}) \cdot \lvert t_1\rvert^{\mathrm{c}} + \mathrm{r}_f(\mathsf{F}) \cdot \lvert t_2\rvert^{\mathrm{c}})$
$\lvert\langle X \vert E\rangle\rvert^{\mathrm{c}} = \langle X \vert \{Y = \lvert t_Y\rvert^{\mathrm{c}} \mid Y = t_Y \in E\}\rangle$

for arbitrary constants of ACP$^\tau$ different from τ and n stands for an arbitrary natural number.

We will write $\sum_{i \in S} p_i$, where $S = \{i_1, \ldots, i_n\}$ and $p_{i_1}, \ldots, p_{i_n}$ are ACP$^\tau$ terms, for $p_{i_1} + \ldots + p_{i_n}$. The convention is that $\sum_{i \in S} p_i$ stands for δ if $S = \emptyset$. We will often write X for $\langle X \vert E\rangle$ if E is clear from the context. It should be borne in mind that, in such cases, we use X as a constant.

4 Process Extraction

In this section, we use ACP$^\tau$ with guarded recursion to make mathematically precise what processes are produced by the threads denoted by closed terms of BTA with guarded recursion.

For that purpose, A and $\mid$ are taken such that the following conditions are satisfied:

$$\mathsf{A} \supseteq \{\mathrm{s}_f(d) \mid f \in \mathcal{F}, d \in \mathcal{M} \cup \mathbb{B}\} \cup \{\mathrm{r}_f(d) \mid f \in \mathcal{F}, d \in \mathcal{M} \cup \mathbb{B}\} \cup \{\mathrm{stop}, \mathrm{i}\}$$

and for all $f \in \mathcal{F}$, $d \in \mathcal{M} \cup \mathbb{B}$, and $e \in \mathsf{A}$:

$$\begin{array}{lll} \mathrm{s}_f(d) \mid \mathrm{r}_f(d) = \mathrm{i}\ , & & \\ \mathrm{s}_f(d) \mid e = \delta & \mathsf{if}\ e \neq \mathrm{r}_f(d)\ , & \mathrm{stop} \mid e = \delta\ , \\ e \mid \mathrm{r}_f(d) = \delta & \mathsf{if}\ e \neq \mathrm{s}_f(d)\ , & \mathrm{i} \mid e = \delta\ . \end{array}$$

The *process extraction* operation $\lvert\_\rvert$ determines, for each closed term p of BTA with guarded recursion, a closed term of ACP$^\tau$ with guarded recursion that denotes the process produced by the thread denoted by p. The process extraction operation $\lvert\_\rvert$ is defined by $\lvert p\rvert = \tau_{\{\mathrm{stop}\}}(\lvert p\rvert^{\mathrm{c}})$, where $\lvert\_\rvert^{\mathrm{c}}$ is defined by the equations given in Table 4 (for $f \in \mathcal{F}$ and $m \in \mathcal{M}$).

Two atomic actions are involved in performing a basic action of the form $f.m$: one for sending a request to process command m to the service named f and another for receiving a reply from that service upon completion of the processing. For each closed term p of BTA with guarded recursion, $\lvert p\rvert^{\mathrm{c}}$ denotes a process that in the event of termination performs a special termination action just before termination. Abstracting from this termination action yields the process denoted by $\lvert p\rvert$. Some atomic actions introduced above are not used in the definition of the process extraction operation for BTA. Those atomic actions are commonly used in the definition of the process extraction operation for extensions of BTA in which operators for thread-service interaction occur, see e.g. [8].

Let p be a closed term of BTA with guarded recursion. Then we say that $|p|$ is the *process produced by* p.

The process extraction operation preserves the axioms of BTA with guarded recursion. Roughly speaking, this means that the translations of these axioms are derivable from the axioms of ACP$^\tau$ with guarded recursion. Before we make this fully precise, we have a closer look at the axioms of BTA with guarded recursion.

A proper axiom is an equation or a conditional equation. In Table 1, we do not find proper axioms. Instead of proper axioms, we find axiom schemas without side conditions and axiom schemas with side conditions. The axioms of BTA with guarded recursion are obtained by replacing each axiom schema by all its instances.

We define a function $|\_|$ from the set of all equations and conditional equations of BTA with guarded recursion to the set of all equations of ACP$^\tau$ with guarded recursion as follows:

$$\begin{aligned} |t_1 = t_2| &= |t_1| = |t_2| \;, \\ |E \Rightarrow t_1 = t_2| &= \{|t'_1| = |t'_2| \mid t'_1 = t'_2 \in E\} \Rightarrow |t_1| = |t_2| \;. \end{aligned}$$

Proposition 1. *Let ϕ be an axiom of* BTA *with guarded recursion. Then $|\phi|$ is derivable from the axioms of* ACP$^\tau$ *with guarded recursion.*

Proof. The proof is trivial. □

Proposition 1 would go through if no abstraction of the above-mentioned special termination action was made. Notice further that ACP$^\tau$ without the silent step constant and the abstraction operator, better known as ACP, would suffice if no abstraction of the special termination action was made.

5 A Simple Protocol

In this section, we consider a very simple transmission protocol for passing instructions from a thread to a remote execution environment.

At the location of the thread concerned, two atomic actions are involved in performing a basic action: one for sending a message containing the basic action via a transmission channel to a receiver at the location of the execution environment and another for receiving a reply via a transmission channel from the receiver upon completion of the processing at the location of the execution environment. The receiver waits until a message containing a basic action can be received. Upon reception of a message containing a basic action $f.m$, the receiver sends a request to process command m to the service named f at the location of the execution environment. Next, the receiver waits until a reply from that service can be received. Upon reception of a reply, the receiver forwards the

Table 5. Process extraction for remotely controlled threads

$\lvert X\rvert_{\mathrm{rct}} = X$
$\lvert \mathsf{S}\rvert_{\mathrm{rct}} = \mathrm{s}_1(\mathsf{stop})$
$\lvert \mathsf{D}\rvert_{\mathrm{rct}} = \mathrm{s}_1(\mathsf{dead})$
$\lvert t_1 \trianglelefteq a \trianglerighteq t_2\rvert_{\mathrm{rct}} = \mathrm{s}_1(a) \cdot (\mathrm{r}_4(\mathsf{T}) \cdot \lvert t_1\rvert_{\mathrm{rct}} + \mathrm{r}_4(\mathsf{F}) \cdot \lvert t_2\rvert_{\mathrm{rct}})$
$\lvert \langle X \vert E\rangle\rvert_{\mathrm{rct}} = \langle X \vert \{Y = \lvert t_Y\rvert_{\mathrm{rct}} \mid Y = t_Y \in E\}\rangle$

reply to the thread. Deadlocking and terminating are treated like performing basic actions.

We write $\mathcal{A}'$ for the set $\mathcal{A} \cup \{\mathsf{stop}, \mathsf{dead}\}$.

For the purpose of describing the very simple transmission protocol outlined above in ACP$^\tau$, A and $\mid$ are taken such that, in addition to the conditions mentioned at the beginning of Section 4, the following conditions are satisfied:

$$\begin{aligned}\mathsf{A} \supseteq\ & \{\mathrm{s}_i(d) \mid i \in \{1,2\}, d \in \mathcal{A}'\} \cup \{\mathrm{r}_i(d) \mid i \in \{1,2\}, d \in \mathcal{A}'\} \\ \cup\ & \{\mathrm{s}_i(r) \mid i \in \{3,4\}, r \in \mathbb{B}\} \cup \{\mathrm{r}_i(r) \mid i \in \{3,4\}, r \in \mathbb{B}\} \cup \{\mathrm{j}\}\end{aligned}$$

and for all $i \in \{1,2\}$, $j \in \{3,4\}$, $d \in \mathcal{A}'$, $r \in \mathbb{B}$, and $e \in \mathsf{A}$:

$$\begin{array}{llll} \mathrm{s}_i(d) \mid \mathrm{r}_i(d) = \mathrm{j}\ , & & \mathrm{s}_j(r) \mid \mathrm{r}_j(r) = \mathrm{j}\ , & \\ \mathrm{s}_i(d) \mid e = \delta & \text{if } e \neq \mathrm{r}_i(d)\ , & \mathrm{s}_j(r) \mid e = \delta & \text{if } e \neq \mathrm{r}_j(r)\ , \\ e \mid \mathrm{r}_i(d) = \delta & \text{if } e \neq \mathrm{s}_i(d)\ , & e \mid \mathrm{r}_j(r) = \delta & \text{if } e \neq \mathrm{s}_j(r)\ , \\ & & & \\ \mathrm{j} \mid e = \delta\ . & & & \end{array}$$

We introduce a process extraction operation $\lvert\_\rvert_{\mathrm{rct}}$ which determines, for each closed term p of BTA with guarded recursion, a closed term of ACP$^\tau$ with guarded recursion that denotes the process produced by the thread denoted by p in the case where the thread is remotely controlled. This operation is defined by the equations given in Table 5 (for $a \in \mathcal{A}$).

Let p be a closed term of BTA with guarded recursion. Then the process representing the remotely controlled thread p is described by

$$\partial_H(\lvert p\rvert_{\mathrm{rct}} \parallel \mathit{CHA} \parallel \mathit{CHR} \parallel \mathit{RCV})\ ,$$

where

$$\begin{aligned} \mathit{CHA} &= \sum_{d \in \mathcal{A}'} \mathrm{r}_1(d) \cdot \mathrm{s}_2(d) \cdot \mathit{CHA}\ , \\ \mathit{CHR} &= \sum_{r \in \mathbb{B}} \mathrm{r}_3(r) \cdot \mathrm{s}_4(r) \cdot \mathit{CHR}\ , \\ \mathit{RCV} &= \sum_{f.m \in \mathcal{A}'} \mathrm{r}_2(f.m) \cdot \mathrm{s}_f(m) \cdot (\mathrm{r}_f(\mathsf{T}) \cdot \mathrm{s}_3(\mathsf{T}) + \mathrm{r}_f(\mathsf{F}) \cdot \mathrm{s}_3(\mathsf{F})) \cdot \mathit{RCV} \\ &\quad + \mathrm{r}_2(\mathsf{stop}) + \mathrm{r}_2(\mathsf{dead}) \cdot \mathrm{i} \cdot \delta \end{aligned}$$

and

$$\begin{aligned} H \;=\; & \{\mathrm{s}_i(d) \mid i \in \{1,2\}\,, d \in \mathcal{A}'\} \cup \{\mathrm{r}_i(d) \mid i \in \{1,2\}\,, d \in \mathcal{A}'\} \\ \cup\; & \{\mathrm{s}_i(r) \mid i \in \{3,4\}\,, r \in \mathbb{B}\} \cup \{\mathrm{r}_i(r) \mid i \in \{3,4\}\,, r \in \mathbb{B}\}\ . \end{aligned}$$

CHA is the transmission channel for messages containing basic actions, *CHR* is the transmission channel for replies, and *RCV* is the receiver.

If we abstract from all atomic actions for sending and receiving via the transmission channels *CHA* and *CHR*, then the processes denoted by $|p|$ and $\partial_H(|p|_{\mathrm{rct}} \parallel \mathit{CHA} \parallel \mathit{CHR} \parallel \mathit{RCV})$ are equal modulo an initial silent step.

Theorem 1. *For each closed term p of* BTA *with guarded recursion:*

$$\tau \cdot |p| = \tau \cdot \tau_{\{\mathrm{j}\}}(\partial_H(|p|_{\mathrm{rct}} \parallel \mathit{CHA} \parallel \mathit{CHR} \parallel \mathit{RCV}))\ .$$

Proof. By AIP, it is sufficient to prove that for all $n \geq 0$:

$$\pi_n(\tau \cdot |p|) = \pi_n(\tau \cdot \tau_{\{\mathrm{j}\}}(\partial_H(|p|_{\mathrm{rct}} \parallel \mathit{CHA} \parallel \mathit{CHR} \parallel \mathit{RCV})))\ .$$

This is easily proved by induction on n and in the inductive step by case distinction on the structure of p, using the axioms of ACP$^\tau$ and RDP. □

6 A More Complex Protocol

In this section, we consider a more complex transmission protocol for passing instructions from a thread to a remote execution environment.

The general idea of this protocol is that:

- while the last basic action performed by the thread in question is processed at the location of the receiver, the first basic actions of the two ways in which the thread may proceed are transmitted together to the receiver;
- while the choice between those two basic actions is made by the receiver on the basis of the reply produced at the completion of the processing, the reply is transferred to the thread.

To simplify the description of the protocol, the following extensions of ACP from [1] will be used:

- We will use conditionals. The expression $p \lhd b \rhd q$, is to be read as `if` b `then` p `else` q. The defining equations are

$$x \lhd \mathsf{T} \rhd y = x \quad \text{and} \quad x \lhd \mathsf{F} \rhd y = y\ .$$

- We will use the generalization of restricted early input action prefixing to process prefixing. Restricted early input action prefixing is defined by the equation $\mathrm{er}_i^D(u)\,;t = \sum_{d \in D} \mathrm{r}_i(d) \cdot t[d/u]$. We use the extension to processes to express binary parallel input: $(\mathrm{er}_i^{D_1}(u_1) \parallel \mathrm{er}_j^{D_2}(u_2))\,;P$. For this particular case, we have the following equation:

$$\begin{aligned} (\mathrm{er}_i^{D_1}(u_1) \parallel \mathrm{er}_j^{D_2}(u_2))\,;t = & \sum_{d_1 \in D_1} \mathrm{r}_i(d_1) \cdot (\mathrm{er}_j^{D_2}(u_2)\,;t[d_1/u_1]) \\ & + \sum_{d_2 \in D_2} \mathrm{r}_j(d_2) \cdot (\mathrm{er}_i^{D_1}(u_1)\,;t[d_2/u_2])\ . \end{aligned}$$

Table 6. Alternative process extraction for remotely controlled threads

$$
\begin{aligned}
&|X|_{\mathrm{rct2}} = X\\
&|\mathsf{S}|_{\mathrm{rct2}} = \mathrm{s}_1(\mathsf{stop})\\
&|\mathsf{D}|_{\mathrm{rct2}} = \mathrm{s}_1(\mathsf{dead})\\
&|t_1 \unlhd a \unrhd t_2|_{\mathrm{rct2}} = \mathrm{s}_1(a, \mathit{init}(t_1), \mathit{init}(t_2)) \cdot (\mathrm{r}_4(\mathsf{T}) \cdot |t_1|'_{\mathrm{rct2}} + \mathrm{r}_4(\mathsf{F}) \cdot |t_2|'_{\mathrm{rct2}})\\
&|\langle X|E\rangle|_{\mathrm{rct2}} = \langle X|\,\{Y = |t_Y|_{\mathrm{rct2}} \mid Y = t_Y \;\in\; E\}\rangle
\end{aligned}
$$

$$
\begin{aligned}
&|X|'_{\mathrm{rct2}} = X\\
&|\mathsf{S}|'_{\mathrm{rct2}} = \mathrm{s}_1(\mathsf{void})\\
&|\mathsf{D}|'_{\mathrm{rct2}} = \mathrm{s}_1(\mathsf{void})\\
&|t_1 \unlhd a \unrhd t_2|'_{\mathrm{rct2}} = \mathrm{s}_1(\mathit{init}(t_1), \mathit{init}(t_2)) \cdot (\mathrm{r}_4(\mathsf{T}) \cdot |t_1|'_{\mathrm{rct2}} + \mathrm{r}_4(\mathsf{F}) \cdot |t_2|'_{\mathrm{rct2}})\\
&|\langle X|E\rangle|'_{\mathrm{rct2}} = \langle X|\,\{Y = |t_Y|'_{\mathrm{rct2}} \mid Y = t_Y \;\in\; E\}\rangle
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{init}(\mathsf{S}) = \mathsf{stop}\\
&\mathit{init}(\mathsf{D}) = \mathsf{dead}\\
&\mathit{init}(t_1 \unlhd a \unrhd t_2) = a\\
&\mathit{init}(\langle X|E\rangle) = \mathit{init}(\langle t_X|E\rangle) \quad \text{if } X = t_X \;\in\; E
\end{aligned}
$$

We write $\mathcal{A}''_2$ for the set $\mathcal{A}' \times \mathcal{A}'$, $\mathcal{A}''_3$ for the set $\mathcal{A} \times \mathcal{A}' \times \mathcal{A}'$, and $\mathcal{A}''$ for the set $\mathcal{A}''_2 \cup \mathcal{A}''_3 \cup \{\mathsf{stop}, \mathsf{dead}, \mathsf{void}\}$.

For the purpose of describing the more complex transmission protocol outlined above in ACP$^\tau$, A and $|$ are taken such that, in addition to the conditions mentioned at the beginning of Section 4, the following conditions are satisfied:

$$
\begin{aligned}
\mathsf{A} \;\supseteq\; &\{\mathrm{s}_i(d) \mid i \in \{1,2\}, d \in \mathcal{A}''\} \cup \{\mathrm{r}_i(d) \mid i \in \{1,2\}, d \in \mathcal{A}''\}\\
\cup\; &\{\mathrm{s}_i(r) \mid i \in \{3,4\}, r \in \mathbb{B}\} \cup \{\mathrm{r}_i(r) \mid i \in \{3,4\}, r \in \mathbb{B}\} \cup \{\mathrm{j}\}
\end{aligned}
$$

and for all $i \in \{1,2\}$, $j \in \{3,4\}$, $d \in \mathcal{A}''$, $r \in \mathbb{B}$, and $e \in \mathsf{A}$:

$$
\begin{array}{llll}
\mathrm{s}_i(d) \mid \mathrm{r}_i(d) = \mathrm{j}\ , & & \mathrm{s}_j(r) \mid \mathrm{r}_j(r) = \mathrm{j}\ , & \\
\mathrm{s}_i(d) \mid e = \delta & \text{if } e \neq \mathrm{r}_i(d)\ , & \mathrm{s}_j(r) \mid e = \delta & \text{if } e \neq \mathrm{r}_j(r)\ ,\\
e \mid \mathrm{r}_i(d) = \delta & \text{if } e \neq \mathrm{s}_i(d)\ , & e \mid \mathrm{r}_j(r) = \delta & \text{if } e \neq \mathrm{s}_j(r)\ ,
\end{array}
$$

$$\mathrm{j} \mid e = \delta\ .$$

We introduce a process extraction operation $|\_|_{\mathrm{rct2}}$ which determines, for each closed term p of BTA with guarded recursion, a closed term of ACP$^\tau$ with guarded recursion that denotes the process produced by the thread denoted by p in the case where the thread is remotely controlled by means of the alternative transmission protocol. This operation is defined by the equations given in Table 6 (for $a \in \mathcal{A}$).

Let p be a closed term of BTA with guarded recursion. Then the process representing the remotely controlled thread p is described by

$$\partial_H(|p|_{\mathrm{rct2}} \parallel \mathit{CHA}_2 \parallel \mathit{CHR} \parallel \mathit{RCV}_2)\ ,$$

where

$$\begin{aligned}
\mathit{CHA}_2 &= \sum_{d \in \mathcal{A}''} \mathrm{r}_1(d) \cdot \mathrm{s}_2(d) \cdot \mathit{CHA}_2 \;, \\
\mathit{CHR} &= \sum_{r \in \mathbb{B}} \mathrm{r}_3(r) \cdot \mathrm{s}_4(r) \cdot \mathit{CHR} \;, \\
\mathit{RCV}_2 &= \sum_{(f.m,a,a') \in \mathcal{A}''_3} \mathrm{r}_2(f.m,a,a') \cdot \mathrm{s}_f(m) \\
&\qquad \cdot (\mathrm{r}_f(\mathsf{T}) \cdot \mathit{RCV}'_2(\mathsf{T},a) + \mathrm{r}_f(\mathsf{F}) \cdot \mathit{RCV}'_2(\mathsf{F},a')) \\
&\quad + \mathrm{r}_2(\mathsf{stop}) + \mathrm{r}_2(\mathsf{dead}) \cdot \mathrm{i} \cdot \delta \;, \\
\mathit{RCV}'_2(r,f.m) &= (\mathrm{s}_3(r) \parallel \mathrm{s}_f(m)) \cdot \mathit{RCV}''_2 \;, \\
\mathit{RCV}'_2(r,\mathsf{stop}) &= \mathrm{r}_2(\mathsf{void}) \;, \\
\mathit{RCV}'_2(r,\mathsf{dead}) &= \mathrm{r}_2(\mathsf{void}) \cdot \mathrm{i} \cdot \delta \;, \\
\mathit{RCV}''_2 &= (\mathrm{er}_2^{\mathcal{A}''_2}(u,v) \parallel \mathrm{er}_f^{\mathbb{B}}(\beta)) \,;\, (\mathit{RCV}'_2(\beta,u) \lhd \beta \rhd \mathit{RCV}'_2(\beta,v))
\end{aligned}$$

and

$$\begin{aligned}
H = {} & \{\mathrm{s}_i(d) \mid i \in \{1,2\}\,, d \in \mathcal{A}''\} \cup \{\mathrm{r}_i(d) \mid i \in \{1,2\}\,, d \in \mathcal{A}''\} \\
\cup {} & \{\mathrm{s}_i(r) \mid i \in \{3,4\}\,, r \in \mathbb{B}\} \cup \{\mathrm{r}_i(r) \mid i \in \{3,4\}\,, r \in \mathbb{B}\} \;.
\end{aligned}$$

Notice that the first cycle of the alternative transmission protocol differs fairly from all subsequent ones. This difference gives rise to a slight complication in the proof of Theorem 2 below.

If we abstract from all atomic actions for sending and receiving via the transmission channels CHA_2 and CHR, then the processes denoted by $|p|$ and $\partial_H(|p|_{\mathrm{rct2}} \parallel \mathit{CHA}_2 \parallel \mathit{CHR} \parallel \mathit{RCV}_2)$ are equal modulo an initial silent step.

Theorem 2. *For each closed term p of* BTA *with guarded recursion:*

$$\tau \cdot |p| = \tau \cdot \tau_{\{\mathrm{j}\}}(\partial_H(|p|_{\mathrm{rct2}} \parallel \mathit{CHA}_2 \parallel \mathit{CHR} \parallel \mathit{RCV}_2)) \;.$$

Proof. By AIP, it is sufficient to prove that for all $n \geq 0$:

$$\pi_n(\tau \cdot |p|) = \pi_n(\tau \cdot \tau_{\{\mathrm{j}\}}(\partial_H(|p|_{\mathrm{rct2}} \parallel \mathit{CHA}_2 \parallel \mathit{CHR} \parallel \mathit{RCV}_2))) \;.$$

For $n = 0, 1, 2$, this is easily proved. For $n \geq 3$, it is easily proved in the cases $p \equiv \mathsf{S}$ and $p \equiv \mathsf{D}$, but in the case $p \equiv p_1 \unlhd f.m \unrhd p_2$ we get:

$$\begin{aligned}
&\tau \cdot \mathrm{s}_f(m) \cdot (\mathrm{r}_f(\mathsf{T}) \cdot \pi_{n-2}(|p_1|) + \mathrm{r}_f(\mathsf{F}) \cdot \pi_{n-2}(|p_2|)) \\
&\quad = \tau \cdot \mathrm{s}_f(m) \\
&\qquad \cdot (\mathrm{r}_f(\mathsf{T}) \cdot \pi_{n-2}(\tau_{\{\mathrm{j}\}}(\partial_H(|p_1|'_{\mathrm{rct2}} \parallel \mathit{CHA}_2 \parallel \mathit{CHR} \parallel \mathit{RCV}'_2(\mathsf{T}, \mathit{init}(p_1))))) \\
&\qquad\quad + \mathrm{r}_f(\mathsf{F}) \cdot \pi_{n-2}(\tau_{\{\mathrm{j}\}}(\partial_H(|p_2|'_{\mathrm{rct2}} \parallel \mathit{CHA}_2 \parallel \mathit{CHR} \parallel \mathit{RCV}'_2(\mathsf{F}, \mathit{init}(p_2)))))) \;.
\end{aligned}$$

We have that

$$\begin{aligned}
&\pi_{n-2}(\tau_{\{\mathrm{j}\}}(\partial_H(|p'|'_{\mathrm{rct2}} \parallel \mathit{CHA}_2 \parallel \mathit{CHR} \parallel \mathit{RCV}'_2(\mathsf{T}, \mathit{init}(p'))))) \\
&\quad = \pi_{n-2}(\tau_{\{\mathrm{j}\}}(\partial_H(|p'|'_{\mathrm{rct2}} \parallel \mathit{CHA}_2 \parallel \mathit{CHR} \parallel \mathit{RCV}'_2(\mathsf{F}, \mathit{init}(p')))))
\end{aligned}$$

in the cases $p' \equiv \mathsf{S}$ and $p' \equiv \mathsf{D}$, but not in the case $p' \equiv p'_1 \trianglelefteq f'.m' \trianglerighteq p'_2$. Therefore, we cannot prove

$$\pi_n(\tau \cdot |p|) = \pi_n(\tau \cdot \tau_{\{\mathrm{j}\}}(\partial_H(|p|_{\mathrm{rct2}} \parallel \mathit{CHA}_2 \parallel \mathit{CHR} \parallel \mathit{RCV}_2)))$$

by induction on n. However, in the case $p' \equiv p'_1 \trianglelefteq f'.m' \trianglerighteq p'_2$ we have that

$$\begin{aligned}&\mathrm{r}_f(r) \cdot \pi_{n-2}(|p'|)\\ &\quad = \mathrm{r}_f(r) \cdot \mathrm{s}_{f'}(m') \cdot \pi_{n-3}(\mathrm{r}_{f'}(\mathsf{T}) \cdot |p'_1| + \mathrm{r}_{f'}(\mathsf{F}) \cdot |p'_2|)\end{aligned}$$

and

$$\begin{aligned}&\mathrm{r}_f(r) \cdot \pi_{n-2}(\tau_{\{\mathrm{j}\}}(\partial_H(|p'|'_{\mathrm{rct2}} \parallel \mathit{CHA}_2 \parallel \mathit{CHR} \parallel \mathit{RCV}'_2(r, f'.m'))))\\ &\quad = \mathrm{r}_f(r) \cdot \mathrm{s}_{f'}(m') \cdot \pi_{n-3}(\tau_{\{\mathrm{j}\}}(\partial_H(|p'|'_{\mathrm{rct2}} \parallel \mathit{CHA}_2 \parallel \mathit{CHR} \parallel \mathit{RCV}''_2))) \;.\end{aligned}$$

Therefore, it is sufficient to prove that for all closed terms p_1 and p_2 of BTA with guarded recursion, $f \in \mathcal{F}$ and $m \in \mathcal{M}$, for all $n \geq 0$:

$$\begin{aligned}&\pi_n(\tau \cdot \ (\mathrm{r}_f(\mathsf{T}) \cdot |p_1| + \mathrm{r}_f(\mathsf{F}) \cdot |p_2|))\\ &\quad = \pi_n(\tau \cdot \tau_{\{\mathrm{j}\}}(\partial_H(|p_1 \trianglelefteq f.m \trianglerighteq p_2|'_{\mathrm{rct2}} \parallel \mathit{CHA}_2 \parallel \mathit{CHR} \parallel \mathit{RCV}''_2))) \;.\end{aligned}$$

This is easily proved by induction on n and in the inductive step by case distinction on the structure of p_1 and p_2, using the axioms of ACP$^\tau$, RDP and the axioms concerning process prefixing and conditionals given in [1]. □

7 Conclusions

Using ACP$^\tau$, we have described a very simple transmission protocol for passing instructions from a thread to a remote execution environment and a more complex one that is more efficient, and we have verified the correctness of these protocols. In this way, we have clarified the phenomenon of remotely controlled program behaviours to a certain extent.

One option for future work is to describe the protocols concerned in a version of ACP with discrete relative timing (see e.g. [2, 3]) and then to show that the more complex one leads to a speed-up indeed. Another option for future work is to devise, describe and analyse more efficient protocols, such as protocols that allow for two or more instructions to be processed in parallel.

By means of the protocols, we have presented a way to deal with the instruction streams that turn up with remotely controlled program behaviours. By that we have ascribed a sense to the term instruction stream which makes clear that an instruction stream is dynamic by nature, in contradistinction with an instruction sequence. We have not yet been able to devise a basic definition of instruction streams.

References

1. Baeten, J.C.M., Bergstra, J.A.: On sequential composition, action prefixes and process prefix. Formal Aspects of Computing **6**(3), 250–268 (1994)
2. Baeten, J.C.M., Bergstra, J.A.: Discrete time process algebra. Formal Aspects of Computing **8**(2), 188–208 (1996)
3. Baeten, J.C.M., Middelburg, C.A.: Process Algebra with Timing. Monographs in Theoretical Computer Science, An EATCS Series. Springer-Verlag, Berlin (2002)
4. Baeten, J.C.M., Weijland, W.P.: Process Algebra, *Cambridge Tracts in Theoretical Computer Science*, vol. 18. Cambridge University Press, Cambridge (1990)
5. Bergstra, J.A., Bethke, I.: Polarized process algebra and program equivalence. In: J.C.M. Baeten, J.K. Lenstra, J. Parrow, G.J. Woeginger (eds.) Proceedings 30th ICALP, *Lecture Notes in Computer Science*, vol. 2719, pp. 1–21. Springer-Verlag (2003)
6. Bergstra, J.A., Klop, J.W.: Process algebra for synchronous communication. Information and Control **60**(1/3), 109–137 (1984)
7. Bergstra, J.A., Loots, M.E.: Program algebra for sequential code. Journal of Logic and Algebraic Programming **51**(2), 125–156 (2002)
8. Bergstra, J.A., Middelburg, C.A.: Thread algebra with multi-level strategies. Fundamenta Informaticae **71**(2/3), 153–182 (2006)
9. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. Journal of the ACM **31**(3), 560–599 (1984)
10. Fokkink, W.J.: Introduction to Process Algebra. Texts in Theoretical Computer Science, An EATCS Series. Springer-Verlag, Berlin (2000)
11. Hennessy, M., Milner, R.: Algebraic laws for non-determinism and concurrency. Journal of the ACM **32**(1), 137–161 (1985)
12. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Englewood Cliffs (1985)
13. Milner, R.: Communication and Concurrency. Prentice-Hall, Englewood Cliffs (1989)

[PRG0709] J.A. Bergstra and C.A. Middelburg, *Instruction Sequences with Indirect Jumps,* Programming Research Group - University of Amsterdam, 2007.

[PRG0708] B. Diertens, *Software (Re-)Engineering with PSF III: an IDE for PSF,* Programming Research Group - University of Amsterdam, 2007.

[PRG0707] J.A. Bergstra and C.A. Middelburg, *An Interface Group for Process Components,* Programming Research Group - University of Amsterdam, 2007.

[PRG0706] J.A. Bergstra, Y. Hirschfeld, and J.V. Tucker, *Skew Meadows,* Programming Research Group - University of Amsterdam, 2007.

[PRG0705] J.A. Bergstra, Y. Hirschfeld, and J.V. Tucker, *Meadows,* Programming Research Group - University of Amsterdam, 2007.

[PRG0704] J.A. Bergstra and C.A. Middelburg, *Machine Structure Oriented Control Code Logic (Extended Version),* Programming Research Group - University of Amsterdam, 2007.

[PRG0703] J.A. Bergstra and C.A. Middelburg, *On the Operating Unit Size of Load/Store Architectures,* Programming Research Group - University of Amsterdam, 2007.

[PRG0702] J.A. Bergstra and A. Ponse, *Interface Groups and Financial Transfer Architectures,* Programming Research Group - University of Amsterdam, 2007.

[PRG0701] J.A. Bergstra, I. Bethke, and M. Burgess, *A Process Algebra Based Framework for Promise Theory,* Programming Research Group - University of Amsterdam, 2007.

[PRG0610] J.A. Bergstra and C.A. Middelburg, *Parallel Processes with Implicit Computational Capital,* Programming Research Group - University of Amsterdam, 2006.

[PRG0609] B. Diertens, *Software (Re-)Engineering with PSF II: from architecture to implementation,* Programming Research Group - University of Amsterdam, 2006.

[PRG0608] A. Ponse and M.B. van der Zwaag, *Risk Assessment for One-Counter Threads,* Programming Research Group - University of Amsterdam, 2006.

[PRG0607] J.A. Bergstra and C.A. Middelburg, *Synchronous Cooperation for Explicit Multi-Threading,* Programming Research Group - University of Amsterdam, 2006.

[PRG0606] J.A. Bergstra and M. Burgess, *Local and Global Trust Based on the Concept of Promises,* Programming Research Group - University of Amsterdam, 2006.

[PRG0605] J.A. Bergstra and J.V. Tucker, *Division Safe Calculation in Totalised Fields,* Programming Research Group - University of Amsterdam, 2006.

[PRG0604] J.A. Bergstra and A. Ponse, *Projection Semantics for Rigid Loops,* Programming Research Group - University of Amsterdam, 2006.

[PRG0603] J.A. Bergstra and I. Bethke, *Predictable and Reliable Program Code: Virtual Machine-based Projection Semantics (submitted for inclusion in the Handbook of Network and Systems Administration),* Programming Research Group - University of Amsterdam, 2006.

[PRG0602] J.A. Bergstra and A. Ponse, *Program Algebra with Repeat Instruction,* Programming Research Group - University of Amsterdam, 2006.

[PRG0601] J.A. Bergstra and A. Ponse, *Interface Groups for Analytic Execution Architectures,* Programming Research Group - University of Amsterdam, 2006.

[PRG0505] B. Diertens, *Software (Re-)Engineering with PSF,* Programming Research Group - University of Amsterdam, 2005.

[PRG0504] P.H. Rodenburg, *Piecewise Initial Algebra Semantics,* Programming Research Group - University of Amsterdam, 2005.

[PRG0503] T.D. Vu, *Metric Denotational Semantics for BPPA,* Programming Research Group - University of Amsterdam, 2005.