Thread algebra with multi-level strategies

Thread Algebra with Multi-Level Strategies

J.A. Bergstra[1,2] and C.A. Middelburg[3]

[1] Programming Research Group, University of Amsterdam,
P.O. Box 41882, 1009 DB Amsterdam, the Netherlands
janb@science.uva.nl
[2] Department of Philosophy, Utrecht University,
P.O. Box 80126, 3508 TC Utrecht, the Netherlands
janb@phil.uu.nl
[3] Computing Science Department, Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, the Netherlands
keesm@win.tue.nl

Abstract. In a previous paper, we developed an algebraic theory of threads and multi-threads based on strategic interleaving. This theory includes a number of plausible interleaving strategies on thread vectors. The strategic interleaving of a thread vector constitutes a multi-thread. Several multi-threads may exist concurrently on a single host in a network, several host behaviors may exist concurrently in a single network on the internet, etc. Strategic interleaving is also present at these other levels. In the current paper, we extend the theory developed so far with features to cover multi-level strategic interleaving. We use the resulting theory to develop a simplified formal representation schema of systems that consist of several multi-threaded programs on various hosts in different networks. We also investigate the connections of the resulting theory with the algebraic theory of processes known as ACP.

Keywords: thread, multi-thread, host, network, service, thread algebra, strategic interleaving, thread-service composition, exception handling, formal design prototype, process algebra.

1 Introduction

A thread is the behavior of a deterministic sequential program under execution. Multi-threading refers to the concurrent existence of several threads in a program under execution. Multi-threading is the dominant form of concurrency provided by recent object-oriented programming languages such as Java [3] and C# [15]. Arbitrary interleaving, on which theories about concurrent processes such as ACP [6] are based, is not the appropriate intuition when dealing with multi-threading. In the case of multi-threading, some deterministic interleaving strategy is used. In [11], we introduced a number of plausible deterministic interleaving strategies for multi-threading. We also proposed to use the phrase strategic interleaving for the more constrained form of interleaving obtained by using such a strategy.

The strategic interleaving of a thread vector constitutes a multi-thread. In conventional operating system jargon, a multi-thread is called a process. Several multi-threads may exist concurrently on the same machine. Multi-processing refers to the concurrent existence of several multi-threads on a machine. Such machines may be hosts in a network, and several host behaviors may exist concurrently in the same network. And so on and so forth. Strategic interleaving is also present at these other levels. In the current paper, we extend the theory developed so far with features to cover multi-level strategic interleaving. There is a dependence on the interleaving strategy considered. We extend the theory only for the simplest case: cyclic interleaving. Other plausible interleaving strategies are treated in [11]. They can also be adapted to the setting of multi-level strategic interleaving.

Threads proceed by performing steps, in the sequel called basic actions, in a sequential fashion. Performing a basic action is taken as making a request to a certain service provided by the execution environment to process a certain command. The service produces a reply value which is returned to the thread concerned. A service may be local to a single thread, local to a multi-thread, local to a host, or local to a network. We introduce thread-service composition in order to bind certain basic actions of a thread to certain services.

An axiomatic description of multi-level strategic interleaving and thread-service composition, as well as a structural operational semantics, is provided. One of our objectives is to develop a simplified, formal representation schema of the design of systems that consist of several multi-threaded programs on various hosts in different networks. We propose to use the term formal design prototype for such a schema. Evidence of the correctness of the presented schema is obtained by a simulation lemma, which states that a finite thread consisting of basic actions that will not be processed by any available service is simulated by any instance of the presented schema that contains the thread in one of its thread vectors.

When a service that is local to a multi-thread receives a request from the multi-thread, it often needs to know from which of the interleaved threads the request originates. This can be achieved by informing the service whenever threads succeed each other by interleaving and whenever a thread drops out by termination or a deadlock. Similar remarks apply to services that are local to hosts and networks. We show how multi-level strategic interleaving can be adapted such that those services are properly informed. We also describe in detail a service that needs such support of thread identity management, using a state-based approach to describe services.

It is interesting to know the connections of threads and services with processes as considered in theories about concurrent processes such as ACP. We show that threads and services can be viewed as processes that are definable over an extension of ACP with conditions introduced in [12] and that thread-service composition on those processes can be expressed in terms of operators of that extension of ACP.

Thread algebra with multi-level strategic interleaving is a design on top of BPPA (Basic Polarized Process Algebra) [8, 5]. BPPA is far less general than ACP-style process algebras and its design focuses on the semantics of deterministic sequential programs. The semantics of a deterministic sequential program is supposed to be a polarized process. Polarization is understood along the axis of the client-server dichotomy. Basic actions in a polarized process are either requests expecting a reply or service offerings promising a reply. Thread algebra may be viewed as client-side polarized process algebra because all threads are viewed as clients generating requests for services provided by their environment.

The structure of this paper is as follows. After a review of BPPA (Section 2), we extend it to a basic thread algebra with cyclic interleaving, but without any feature for multi-level strategic interleaving (Section 3). Next, we extend this basic thread algebra with thread-service composition (Section 4) and other features for multi-level strategic interleaving (Section 5). Following this, we discuss how two additional features can be expressed (Section 6) and give a formal representation schema of the design of systems that consist of several multi-threaded programs on various hosts in different networks (Section 7). Then, we enhance multi-level strategic interleaving with support of thread identity management by services (Section 8). Thereupon, we introduce a state-based approach to describe services (Section 9) and use it to describe a service in which thread identity management is needed (Section 10). After that, we review an extension of ACP with conditions introduced in [12] (Section 11) and show the connections of threads and services with processes that are definable over this extension of ACP (Section 12). Finally, we make some concluding remarks (Section 13).

This paper is a revision and extension of [9].

2 Basic Polarized Process Algebra

In this section, we review BPPA (Basic Polarized Process Algebra), a form of process algebra which is tailored to the use for the description of the behavior of deterministic sequential programs under execution.

In BPPA, it is assumed that there is a fixed but arbitrary finite set of *basic actions* $\mathcal{A}$ with $\mathsf{tau} \notin \mathcal{A}$. We write $\mathcal{A}_{\mathsf{tau}}$ for $\mathcal{A} \cup \{\mathsf{tau}\}$. BPPA has the following constants and operators:

- the *deadlock* constant D;
- the *termination* constant S;
- for each $a \in \mathcal{A}_{\mathsf{tau}}$, a binary *postconditional composition* operator $\_ \trianglelefteq a \trianglerighteq \_$.

We use infix notation for postconditional composition. We introduce *action prefixing* as an abbreviation: $a \circ p$, where p is a term of BPPA, abbreviates $p \trianglelefteq a \trianglerighteq p$.

The intuition is that each basic action is taken as a command to be processed by the execution environment. The processing of a command may involve a change of state of the execution environment. At completion of the processing of the command, the execution environment produces a reply value. This reply is either T or F and is returned to the polarized process concerned. Let p and

Table 1. Axiom of BPPA

$x \trianglelefteq \mathsf{tau} \trianglerighteq y = x \trianglelefteq \mathsf{tau} \trianglerighteq x$	T1

Table 2. Axioms for projection

$\pi_0(x) = \mathsf{D}$	P0
$\pi_{n+1}(\mathsf{S}) = \mathsf{S}$	P1
$\pi_{n+1}(\mathsf{D}) = \mathsf{D}$	P2
$\pi_{n+1}(x \trianglelefteq a \trianglerighteq y) = \pi_n(x) \trianglelefteq a \trianglerighteq \pi_n(y)$	P3
$(\bigwedge_{n \geq 0} \pi_n(x) = \pi_n(y)) \Rightarrow x = y$	AIP

q be closed terms of BPPA. Then $p \trianglelefteq a \trianglerighteq q$ will proceed as p if the processing of a leads to the reply T (called a positive reply), and it will proceed as q if the processing of a leads to the reply F (called a negative reply). If the reply is used to indicate whether the processing was successful, a useful convention is to indicate successful processing by the reply T and unsuccessful processing by the reply F. The action tau plays a special role. Its execution will never change any state and always produces a positive reply.

BPPA has only one axiom. This axiom is given in Table 1. Using the abbreviation introduced above, axiom T1 can be written as follows: $x \trianglelefteq \mathsf{tau} \trianglerighteq y = \mathsf{tau} \circ x$.

A *recursive specification* over BPPA is a set of equations $E = \{X = t_X \mid X \in V\}$, where V is a set of variables and each t_X is a term of BPPA that only contains variables from V. Let t be a term of BPPA containing a variable X. Then an occurrence of X in t is *guarded* if t has a subterm of the form $t' \trianglelefteq a \trianglerighteq t''$ containing this occurrence of X. A recursive specification over BPPA is *guarded* if all occurrences of variables in the right-hand sides of its equations are guarded or it can be rewritten to such a recursive specification using the equations of the recursive specification. Following [5], a CPO structure can be imposed on the domain of the projective limit model of BPPA. Then guarded recursive specifications represent continuous operators having least fixed points. These matters will not be repeated here, taking for granted that guarded recursive specifications over BPPA have unique solutions. For each guarded recursive specification E over BPPA and each variable X that occurs as the left-hand side of an equation in E, we add to the constants of BPPA a constant standing for the unique solution of E for X. This constant is denoted by $\langle X|E \rangle$.

The projective limit characterization of process equivalence on polarized processes is based on the notion of a finite approximation of depth n. When for all n these approximations are identical for two given polarized processes, both processes are considered identical. This allows one to eliminate recursion in favor of the infinitary proof rule AIP. Following [8], which in fact uses the notation of [6], approximation of depth n is phrased in terms of a unary *projection* operator $\pi_n(\_)$. The projection operators are defined inductively by means of the axioms in Table 2. In this table and all subsequent tables with axioms in which

a occurs, a stands for an arbitrary action from $\mathcal{A}_{\mathsf{tau}}$.

As mentioned above, the behavior of a polarized process depends upon its execution environment. Each basic action performed by the polarized process is taken as a command to be processed by the execution environment. At any stage, the commands that the execution environment can accept depend only on its history, i.e. the sequence of commands processed before and the sequence of replies produced for those commands. When the execution environment accepts a command, it will produce a positive reply or a negative reply. Whether the reply is positive or negative usually depends on the execution history. However, it may also depend on external conditions.

In the structural operational semantics, we represent an execution environment by a function $\rho : (\mathcal{A} \times \{\mathsf{T}, \mathsf{F}\})^* \to \mathcal{P}(\mathcal{A} \times \{\mathsf{T}, \mathsf{F}\})$ that satisfies the following condition: $(a, b) \notin \rho(\alpha) \Rightarrow \rho(\alpha \frown \langle (a, b) \rangle) = \emptyset$ for all $a \in \mathcal{A}$, $b \in \{\mathsf{T}, \mathsf{F}\}$ and $\alpha \in (\mathcal{A} \times \{\mathsf{T}, \mathsf{F}\})^*$.[4] We write $\mathcal{E}$ for the set of all those functions. Given an execution environment $\rho \in \mathcal{E}$ and a basic action $a \in \mathcal{A}$, the *derived* execution environment of ρ after processing a with a *positive* reply, written $\frac{\partial^+}{\partial a} \rho$, is defined by $\frac{\partial^+}{\partial a} \rho(\alpha) = \rho(\langle (a, \mathsf{T}) \rangle \frown \alpha)$; and the *derived* execution environment of ρ after processing a with a *negative* reply, written $\frac{\partial^-}{\partial a} \rho$, is defined by $\frac{\partial^-}{\partial a} \rho(\alpha) = \rho(\langle (a, \mathsf{F}) \rangle \frown \alpha)$.

The following transition relations on closed terms are used in the structural operational semantics of BPPA:

- a binary relation $\langle \_\,, \rho \rangle \xrightarrow{a} \langle \_\,, \rho' \rangle$ for each $a \in \mathcal{A}_{\mathsf{tau}}$ and $\rho, \rho' \in \mathcal{E}$;
- a unary relation $\langle \_\,, \rho \rangle\!\downarrow$ for each $\rho \in \mathcal{E}$;
- a unary relation $\langle \_\,, \rho \rangle\!\uparrow$ for each $\rho \in \mathcal{E}$.

The three kinds of transition relations are called the *action step*, *termination*, and *deadlock* relations, respectively. They can be explained as follows:

- $\langle p, \rho \rangle \xrightarrow{a} \langle p', \rho' \rangle$: in execution environment ρ, process p is capable of first performing action a and then proceeding as process p' in execution environment ρ';
- $\langle p, \rho \rangle\!\downarrow$: in execution environment ρ, process p is capable of terminating successfully;
- $\langle p, \rho \rangle\!\uparrow$: in execution environment ρ, process p is neither capable of performing an action nor capable of terminating successfully.

The structural operational semantics of BPPA extended with projection and recursion is described by the transition rules given in Table 3. In this table and all subsequent tables with transition rules in which a occurs, a stands for an arbitrary action from $\mathcal{A}_{\mathsf{tau}}$. We write $\langle t | E \rangle$ for t with, for all X that occur on the left-hand side of an equation in E, all occurrences of X in t replaced by $\langle X | E \rangle$.

Bisimulation equivalence is defined as follows. A *bisimulation* is a symmetric binary relation B on closed terms such that for all closed terms p and q:

[4] We write $\langle\,\rangle$ for the empty sequence, $\langle d \rangle$ for the sequence having d as sole element, and $\alpha \frown \beta$ for the concatenation of sequences α and β. We assume that the identities $\alpha \frown \langle\,\rangle = \langle\,\rangle \frown \alpha = \alpha$ hold.

Table 3. Transition rules for BPPA with projection and recursion

$$\frac{}{\langle \mathsf{S},\rho\rangle{\downarrow}} \qquad \frac{}{\langle \mathsf{D},\rho\rangle{\uparrow}}$$

$$\frac{}{\langle x \unlhd a \unrhd y,\rho\rangle \xrightarrow{a} \langle x, \frac{\partial^+}{\partial a}\rho\rangle}\ (a,\mathsf{T})\in\rho(\langle\,\rangle) \qquad \frac{}{\langle x \unlhd a \unrhd y,\rho\rangle \xrightarrow{a} \langle y, \frac{\partial^-}{\partial a}\rho\rangle}\ (a,\mathsf{F})\in\rho(\langle\,\rangle)$$

$$\frac{}{\langle x \unlhd a \unrhd y,\rho\rangle{\uparrow}}\ (a,\mathsf{T})\notin\rho(\langle\,\rangle),\ (a,\mathsf{F})\notin\rho(\langle\,\rangle) \qquad \frac{}{\langle x \unlhd \mathsf{tau} \unrhd y,\rho\rangle \xrightarrow{\mathsf{tau}} \langle x,\rho\rangle}$$

$$\frac{\langle x,\rho\rangle \xrightarrow{a} \langle x',\rho'\rangle}{\langle \pi_{n+1}(x),\rho\rangle \xrightarrow{a} \langle \pi_n(x'),\rho'\rangle} \qquad \frac{\langle x,\rho\rangle{\downarrow}}{\langle \pi_{n+1}(x),\rho\rangle{\downarrow}} \qquad \frac{\langle x,\rho\rangle{\uparrow}}{\langle \pi_{n+1}(x),\rho\rangle{\uparrow}} \qquad \frac{}{\langle \pi_0(x),\rho\rangle{\uparrow}}$$

$$\frac{\langle\langle t|E\rangle,\rho\rangle \xrightarrow{a} \langle x',\rho'\rangle}{\langle\langle X|E\rangle,\rho\rangle \xrightarrow{a} \langle x',\rho'\rangle}\ X{=}t\in E \qquad \frac{\langle\langle t|E\rangle,\rho\rangle{\downarrow}}{\langle\langle X|E\rangle,\rho\rangle{\downarrow}}\ X{=}t\in E \qquad \frac{\langle\langle t|E\rangle,\rho\rangle{\uparrow}}{\langle\langle X|E\rangle,\rho\rangle{\uparrow}}\ X{=}t\in E$$

- if $B(p,q)$ and $\langle p,\rho\rangle \xrightarrow{a} \langle p',\rho'\rangle$, then there is a q' such that $\langle q,\rho\rangle \xrightarrow{a} \langle q',\rho'\rangle$ and $B(p',q')$;
- if $B(p,q)$ and $\langle p,\rho\rangle{\downarrow}$, then $\langle q,\rho\rangle{\downarrow}$;
- if $B(p,q)$ and $\langle p,\rho\rangle{\uparrow}$, then $\langle q,\rho\rangle{\uparrow}$.

Two closed terms p and q are *bisimulation equivalent*, written $p \leftrightarrows q$, if there exists a bisimulation B such that $B(p,q)$.

Bisimulation equivalence is a congruence with respect to the postconditional composition operators and the projection operators. This follows immediately from the fact that the transition rules for BPPA with projection and recursion constitute a transition system specification in path format (see e.g. [2]).

3 Basic Thread Algebra with Foci and Methods

In this section, we introduce a thread algebra without features for multi-level strategic interleaving. Such features will be added in subsequent sections. It is a design on top of BPPA.

In [8], its has been outlined how and why polarized processes are a natural candidate for the specification of the semantics of deterministic sequential programs. Assuming that a thread is a process representing a deterministic sequential program under execution, it is reasonable to view all polarized processes as threads. A thread vector is a sequence of threads.

Strategic interleaving operators turn a thread vector of arbitrary length into a single thread. This single thread obtained via a strategic interleaving operator is also called a multi-thread. Formally, however both threads and multi-threads are polarized processes. In this paper, we only cover the simplest interleaving strategy, namely *cyclic interleaving*. Other plausible interleaving strategies are treated in [11]. They can also be adapted to the features for multi-level strategic interleaving that will be introduced in the current paper. The strategic interleav-

Table 4. Axioms for cyclic interleaving

$\|(\langle\,\rangle) = \mathsf{S}$	CSI1
$\|(\langle \mathsf{S}\rangle \frown \alpha) = \|(\alpha)$	CSI2
$\|(\langle \mathsf{D}\rangle \frown \alpha) = \mathsf{S}_{\mathsf{D}}(\|(\alpha))$	CSI3
$\|(\langle \mathsf{tau} \circ x\rangle \frown \alpha) = \mathsf{tau} \circ \|(\alpha \frown \langle x\rangle)$	CSI4
$\|(\langle x \unlhd f.m \unrhd y\rangle \frown \alpha) = \|(\alpha \frown \langle x\rangle) \unlhd f.m \unrhd \|(\alpha \frown \langle y\rangle)$	CSI5

Table 5. Axioms for deadlock at termination

$\mathsf{S}_{\mathsf{D}}(\mathsf{S}) = \mathsf{D}$	S2D1
$\mathsf{S}_{\mathsf{D}}(\mathsf{D}) = \mathsf{D}$	S2D2
$\mathsf{S}_{\mathsf{D}}(\mathsf{tau} \circ x) = \mathsf{tau} \circ \mathsf{S}_{\mathsf{D}}(x)$	S2D3
$\mathsf{S}_{\mathsf{D}}(x \unlhd f.m \unrhd y) = \mathsf{S}_{\mathsf{D}}(x) \unlhd f.m \unrhd \mathsf{S}_{\mathsf{D}}(y)$	S2D4

Table 6. Transition rules for cyclic interleaving and deadlock at termination

$$\frac{\langle x_1,\rho\rangle{\downarrow},\ldots,\langle x_k,\rho\rangle{\downarrow},\langle x_{k+1},\rho\rangle \xrightarrow{a} \langle x'_{k+1},\rho'\rangle}{\langle \|(\langle x_1\rangle \frown \ldots \frown \langle x_{k+1}\rangle \frown \alpha),\rho\rangle \xrightarrow{a} \langle \|(\alpha \frown \langle x'_{k+1}\rangle),\rho'\rangle} \quad (k \geq 0)$$

$$\frac{\langle x_1,\rho\rangle \not\rightarrow,\ldots,\langle x_k,\rho\rangle \not\rightarrow,\langle x_l,\rho\rangle{\uparrow},\langle x_{k+1},\rho\rangle \xrightarrow{a} \langle x'_{k+1},\rho'\rangle}{\langle \|(\langle x_1\rangle \frown \ldots \frown \langle x_{k+1}\rangle \frown \alpha),\rho\rangle \xrightarrow{a} \langle \|(\alpha \frown \langle \mathsf{D}\rangle \frown \langle x'_{k+1}\rangle),\rho'\rangle} \quad (k \geq l > 0)$$

$$\frac{\langle x_1,\rho\rangle{\downarrow},\ldots,\langle x_k,\rho\rangle{\downarrow}}{\langle \|(\langle x_1\rangle \frown \ldots \frown \langle x_k\rangle),\rho\rangle{\downarrow}} \qquad \frac{\langle x_1,\rho\rangle \not\rightarrow,\ldots,\langle x_k,\rho\rangle \not\rightarrow,\langle x_l,\rho\rangle{\uparrow}}{\langle \|(\langle x_1\rangle \frown \ldots \frown \langle x_k\rangle),\rho\rangle{\uparrow}} \quad (k \geq l > 0)$$

$$\frac{\langle x,\rho\rangle \xrightarrow{a} \langle x',\rho'\rangle}{\langle \mathsf{S}_{\mathsf{D}}(x),\rho\rangle \xrightarrow{a} \langle \mathsf{S}_{\mathsf{D}}(x'),\rho'\rangle} \qquad \frac{\langle x,\rho\rangle{\downarrow}}{\langle \mathsf{S}_{\mathsf{D}}(x),\rho\rangle{\uparrow}} \qquad \frac{\langle x,\rho\rangle{\uparrow}}{\langle \mathsf{S}_{\mathsf{D}}(x),\rho\rangle{\uparrow}}$$

ing operator for cyclic interleaving is denoted by $\|(\_)$. In [11], it was denoted by $\|_{csi}(\_)$ to distinguish it from other strategic interleaving operators.

It is assumed that there is a fixed but arbitrary finite set of *foci* $\mathcal{F}$ and a fixed but arbitrary finite set of *methods* $\mathcal{M}$. For the set of basic actions $\mathcal{A}$, we take the set $\{f.m \mid f \in \mathcal{F}, m \in \mathcal{M}\}$. Each focus plays the role of a name of a service provided by the execution environment that can be requested to process a command. Each method plays the role of a command proper. Performing a basic action $f.m$ is taken as making a request to the service named f to process the command m.

The axioms for cyclic interleaving are given in Table 4. In this table and all subsequent tables with axioms or transition rules in which f and m occur, f and m stand for an arbitrary focus from $\mathcal{F}$ and an arbitrary method from $\mathcal{M}$, respectively. In CSI3, the auxiliary *deadlock at termination* operator $\mathsf{S}_{\mathsf{D}}(\_)$ is used. It turns termination into deadlock. Its axioms appear in Table 5.

The structural operational semantics of the basic thread algebra with foci and methods is described by the transition rules given in Tables 3 and 6. Here $\langle x,\rho\rangle \not\rightarrow$

stands for the set of all negative conditions $\neg\,(\langle x, \rho\rangle \xrightarrow{a} \langle p', \rho'\rangle)$ where p' is a closed term of BPPA, $\rho' \in \mathcal{E}$, $a \in \mathcal{A}_{\mathsf{tau}}$. Recall that $\mathcal{A} = \{f.m \mid f \in \mathcal{F}, m \in \mathcal{M}\}$.

Bisimulation equivalence is also a congruence with respect to the cyclic interleaving operator and the deadlock at termination operator. This follows immediately from the fact that the transition rules for the basic thread algebra with foci and methods constitute a complete transition system specification in relaxed panth format (see e.g. [17]).

4 Thread-Service Composition

In this section, we extend the basic thread algebra with foci and methods with thread-service composition. For each $f \in \mathcal{F}$, we introduce a *thread-service composition* operator $\_ \;/_f\; \_$. These operators have a thread as first argument and a service as second argument. $P \;/_f\; H$ is the thread that results from issuing all basic actions from thread P that are of the form $f.m$ to service H.

A service is represented by a function $H : \mathcal{M}^+ \to \{\mathsf{T}, \mathsf{F}, \mathsf{B}, \mathsf{R}\}$ with the property that $H(\alpha) = \mathsf{B} \Rightarrow H(\alpha \frown \langle m\rangle) = \mathsf{B}$ and $H(\alpha) = \mathsf{R} \Rightarrow H(\alpha \frown \langle m\rangle) = \mathsf{R}$ for all $\alpha \in \mathcal{M}^+$ and $m \in \mathcal{M}$. This function is called the *reply* function of the service. We write $\mathcal{RF}$ for the set of all reply functions and $\mathcal{R}$ for the set $\{\mathsf{T}, \mathsf{F}, \mathsf{B}, \mathsf{R}\}$. Given a reply function H and a method m, the derived reply function of H after processing m, written $\frac{\partial}{\partial m}H$, is defined by $\frac{\partial}{\partial m}H(\alpha) = H(\langle m\rangle \frown \alpha)$.

The connection between a reply function H and the service represented by it can be understood as follows:

- If $H(\langle m\rangle) = \mathsf{T}$, the request to process command m is accepted by the service, the reply is positive and the service proceeds as $\frac{\partial}{\partial m}H$.
- If $H(\langle m\rangle) = \mathsf{F}$, the request to process command m is accepted by the service, the reply is negative and the service proceeds as $\frac{\partial}{\partial m}H$.
- If $H(\langle m\rangle) = \mathsf{B}$, the request to process command m is not refused by the service, but the processing of m is temporarily blocked. The request will have to wait until the processing of m is not blocked any longer.
- If $H(\langle m\rangle) = \mathsf{R}$, the request to process command m is refused by the service.

Henceforth, we will identify a reply function with the service represented by it.

The axioms for thread-service composition are given in Table 7. In this table and all subsequent tables with axioms or transition rules in which g occurs, like f, g stands for an arbitrary focus from $\mathcal{F}$. Moreover, in this table and all subsequent tables with axioms or transition rules in which H occurs, H stands for an arbitrary reply function from $\mathcal{RF}$.

The structural operational semantics of the basic thread algebra with foci and methods extended with thread-service composition is described by the transition rules given in Tables 3, 6 and 8.

The action `tau` arises as the residue of processing commands. Therefore, `tau` is not connected to a particular focus, and is always accepted.

Table 7. Axioms for thread-service composition

$\mathsf{S} /_f H = \mathsf{S}$		TSC1
$\mathsf{D} /_f H = \mathsf{D}$		TSC2
$(\mathsf{tau} \circ x) /_f H = \mathsf{tau} \circ (x /_f H)$		TSC3
$(x \trianglelefteq g.m \trianglerighteq y) /_f H = (x /_f H) \trianglelefteq g.m \trianglerighteq (y /_f H)$	if $\neg\, f = g$	TSC4
$(x \trianglelefteq f.m \trianglerighteq y) /_f H = \mathsf{tau} \circ (x /_f \frac{\partial}{\partial m} H)$	if $H(\langle m \rangle) = \mathsf{T}$	TSC5
$(x \trianglelefteq f.m \trianglerighteq y) /_f H = \mathsf{tau} \circ (y /_f \frac{\partial}{\partial m} H)$	if $H(\langle m \rangle) = \mathsf{F}$	TSC6
$(x \trianglelefteq f.m \trianglerighteq y) /_f H = \mathsf{D}$	if $H(\langle m \rangle) = \mathsf{B} \lor H(\langle m \rangle) = \mathsf{R}$	TSC7

Table 8. Transition rules for thread-service composition

$$\frac{\langle x, \rho \rangle \xrightarrow{g.m} \langle x', \rho' \rangle}{\langle x /_f H, \rho \rangle \xrightarrow{g.m} \langle x' /_f H, \rho' \rangle}\; f \neq g \qquad \frac{\langle x, \rho \rangle \xrightarrow{\mathsf{tau}} \langle x', \rho' \rangle}{\langle x /_f H, \rho \rangle \xrightarrow{\mathsf{tau}} \langle x' /_f H, \rho' \rangle}$$

$$\frac{\langle x, \rho \rangle \xrightarrow{f.m} \langle x', \rho' \rangle}{\langle x /_f H, \rho \rangle \xrightarrow{\mathsf{tau}} \langle x' /_f \frac{\partial}{\partial m} H, \rho' \rangle}\; H(\langle m \rangle) \in \{\mathsf{T}, \mathsf{F}\},\ (f.m, H(\langle m \rangle)) \in \rho(\langle\,\rangle)$$

$$\frac{\langle x, \rho \rangle \xrightarrow{f.m} \langle x', \rho' \rangle}{\langle x /_f H, \rho \rangle \uparrow}\; H(\langle m \rangle) \in \{\mathsf{B}, \mathsf{R}\} \qquad \frac{\langle x, \rho \rangle \downarrow}{\langle x /_f H, \rho \rangle \downarrow} \qquad \frac{\langle x, \rho \rangle \uparrow}{\langle x /_f H, \rho \rangle \uparrow}$$

5 Guarding Tests

In this section, we extend the thread algebra developed so far with guarding tests. Guarding tests are basic actions meant to verify whether a service will accept the request to process a certain method now, and if not so whether it will be accepted after some time. Guarding tests allow for dealing with delayed processing and exception handling as will be shown in Section 6.

We extend the set of basic actions. For the set of basic actions, we now take the set $\{f.m, f?m, f??m \mid f \in \mathcal{F}, m \in \mathcal{M}\}$. Basic actions of the forms $f?m$ and $f??m$ will be called *guarding tests*. Performing a basic action $f?m$ is taken as making the request to the service named f to reply whether it will accept the request to process method m now. The reply is positive if the service will accept that request now, and otherwise it is negative. Performing a basic action $f??m$ is taken as making the request to the service named f to reply whether it will accept the request to process method m now or after some time. The reply is positive if the service will accept that request now or after some time, and otherwise it is negative.

As explained below, it happens that not only thread-service composition but also cyclic interleaving has to be adapted to the presence of guarding tests.

The additional axioms for cyclic interleaving and deadlock at termination in the presence of guarding tests are given in Table 9. Axioms CSI6 and CSI7 state that:

- after a positive reply on $f?m$ or $f??m$, the same thread proceeds with its next basic action; and thus it is prevented that meanwhile other threads can

Table 9. Additional axioms for cyclic interleaving & deadlock at termination

$\parallel(\langle x \unlhd f?m \unrhd y\rangle \curvearrowright \alpha) = \parallel(\langle x\rangle \curvearrowright \alpha) \unlhd f?m \unrhd \parallel(\alpha \curvearrowright \langle y\rangle)$	CSI6
$\parallel(\langle x \unlhd f??m \unrhd y\rangle \curvearrowright \alpha) = \parallel(\langle x\rangle \curvearrowright \alpha) \unlhd f??m \unrhd \parallel(\alpha \curvearrowright \langle y\rangle)$	CSI7
$\mathsf{S}_{\mathsf{D}}(x \unlhd f?m \unrhd y) = \mathsf{S}_{\mathsf{D}}(x) \unlhd f?m \unrhd \mathsf{S}_{\mathsf{D}}(y)$	S2D5
$\mathsf{S}_{\mathsf{D}}(x \unlhd f??m \unrhd y) = \mathsf{S}_{\mathsf{D}}(x) \unlhd f??m \unrhd \mathsf{S}_{\mathsf{D}}(y)$	S2D6

Table 10. Additional transition rules for cyclic interleaving & deadlock at termination

$$\frac{\langle x_1,\rho\rangle{\downarrow},\ldots,\langle x_k,\rho\rangle{\downarrow},\langle x_{k+1},\rho\rangle \xrightarrow{\gamma} \langle x'_{k+1},\rho'\rangle}{\langle \parallel(\langle x_1\rangle \curvearrowright \ldots \curvearrowright \langle x_{k+1}\rangle \curvearrowright \alpha),\rho\rangle \xrightarrow{\gamma} \langle \parallel(\langle x'_{k+1}\rangle \curvearrowright \alpha),\rho'\rangle}\ (\alpha,\mathsf{T}) \in \rho(\langle\,\rangle) \qquad (k \geq 0)$$

$$\frac{\langle x_1,\rho\rangle\not\rightarrow,\ldots,\langle x_k,\rho\rangle\not\rightarrow,\langle x_l,\rho\rangle{\uparrow},\langle x_{k+1},\rho\rangle \xrightarrow{\gamma} \langle x'_{k+1},\rho'\rangle}{\langle \parallel(\langle x_1\rangle \curvearrowright \ldots \curvearrowright \langle x_{k+1}\rangle \curvearrowright \alpha),\rho\rangle \xrightarrow{\gamma} \langle \parallel(\langle x'_{k+1}\rangle \curvearrowright \alpha \curvearrowright \langle \mathsf{D}\rangle),\rho'\rangle}\ (\alpha,\mathsf{T}) \in \rho(\langle\,\rangle) \quad (k \geq l > 0)$$

$$\frac{\langle x_1,\rho\rangle{\downarrow},\ldots,\langle x_k,\rho\rangle{\downarrow},\langle x_{k+1},\rho\rangle \xrightarrow{\gamma} \langle x'_{k+1},\rho'\rangle}{\langle \parallel(\langle x_1\rangle \curvearrowright \ldots \curvearrowright \langle x_{k+1}\rangle \curvearrowright \alpha),\rho\rangle \xrightarrow{\gamma} \langle \parallel(\alpha \curvearrowright \langle x'_{k+1}\rangle),\rho'\rangle}\ (\alpha,\mathsf{F}) \in \rho(\langle\,\rangle) \qquad (k \geq 0)$$

$$\frac{\langle x_1,\rho\rangle\not\rightarrow,\ldots,\langle x_k,\rho\rangle\not\rightarrow,\langle x_l,\rho\rangle{\uparrow},\langle x_{k+1},\rho\rangle \xrightarrow{\gamma} \langle x'_{k+1},\rho'\rangle}{\langle \parallel(\langle x_1\rangle \curvearrowright \ldots \curvearrowright \langle x_{k+1}\rangle \curvearrowright \alpha),\rho\rangle \xrightarrow{\gamma} \langle \parallel(\alpha \curvearrowright \langle \mathsf{D}\rangle \curvearrowright \langle x'_{k+1}\rangle),\rho'\rangle}\ (\alpha,\mathsf{F}) \in \rho(\langle\,\rangle) \quad (k \geq l > 0)$$

$$\frac{\langle x,\rho\rangle \xrightarrow{\gamma} \langle x',\rho'\rangle}{\langle \mathsf{S}_{\mathsf{D}}(x),\rho\rangle \xrightarrow{\gamma} \langle \mathsf{S}_{\mathsf{D}}(x'),\rho'\rangle}$$

cause a state change to a state in which the processing of m is blocked (and $f?m$ would not reply positively) or the processing of m is refused (and both $f?m$ and $f??m$ would not reply positively);

- after a negative reply on $f?m$ or $f??m$, the same thread does not proceed with it; and thus it is prevented that other threads cannot make progress.

Without this difference, the Simulation Lemma (Section 7) would not go through.

The additional transition rules for cyclic interleaving and deadlock at termination in the presence of guarding tests are given in Table 10, where γ stands for an arbitrary basic action from the set $\{f?m, f??m \mid f \in \mathcal{F}, m \in \mathcal{M}\}$.

A service may be local to a single thread, local to a multi-thread, local to a host, or local to a network. A service local to a multi-thread is shared by all threads from which the multi-thread is composed, etc. Henceforth, to simplify matters, it is assumed that each thread, each multi-thread, each host, and each network has a unique local service. Moreover, it is assumed that $\mathsf{t},\mathsf{p},\mathsf{h},\mathsf{n} \in \mathcal{F}$. Below, the foci t, p, h and n play a special role:

- for each thread, t is the focus of its unique local service;
- for each multi-thread, p is the focus of its unique local service;
- for each host, h is the focus of its unique local service;
- for each network, n is the focus of its unique local service.

The additional axioms for thread-service composition in the presence of guarding tests are given in Table 11. Axioms TSC10 and TSC11 are crucial.

Table 11. Additional axioms for thread-service composition

$(x \unlhd g?m \unrhd y) /_f H = (x /_f H) \unlhd g?m \unrhd (y /_f H)$	if $\neg\, f = g$	TSC8
$(x \unlhd f?m \unrhd y) /_f H = \mathsf{tau} \circ (x /_f H)$	if $H(\langle m \rangle) = \mathsf{T}\ \lor$ $\ \ H(\langle m \rangle) = \mathsf{F}$	TSC9
$(x \unlhd f?m \unrhd y) /_f H = \mathsf{tau} \circ (y /_f H)$	if $H(\langle m \rangle) = \mathsf{B}\ \land\ \neg\, f = \mathsf{t}$	TSC10
$(x \unlhd f?m \unrhd y) /_f H = \mathsf{D}$	if $(H(\langle m \rangle) = \mathsf{B}\ \land\ f = \mathsf{t})\ \lor$ $\ \ H(\langle m \rangle) = \mathsf{R}$	TSC11
$(x \unlhd g??m \unrhd y) /_f H = (x /_f H) \unlhd g??m \unrhd (y /_f H)$	if $\neg\, f = g$	TSC12
$(x \unlhd f??m \unrhd y) /_f H = \mathsf{tau} \circ (x /_f H)$	if $\neg\, H(\langle m \rangle) = \mathsf{R}$	TSC13
$(x \unlhd f??m \unrhd y) /_f H = \mathsf{tau} \circ (y /_f H)$	if $H(\langle m \rangle) = \mathsf{R}$	TSC14

Table 12. Additional transition rules for thread-service composition

$$\frac{\langle x, \rho \rangle \xrightarrow{f?m} \langle x', \rho' \rangle}{\langle x /_f H, \rho \rangle \xrightarrow{\mathsf{tau}} \langle x' /_f H, \rho' \rangle}\ H(\langle m \rangle) \in \{\mathsf{T}, \mathsf{F}\},\ (f?m, \mathsf{T}) \in \rho(\langle\,\rangle)$$

$$\frac{\langle x, \rho \rangle \xrightarrow{f?m} \langle x', \rho' \rangle}{\langle x /_f H, \rho \rangle \xrightarrow{\mathsf{tau}} \langle x' /_f H, \rho' \rangle}\ H(\langle m \rangle) = \mathsf{B},\ f \neq \mathsf{t},\ (f?m, \mathsf{F}) \in \rho(\langle\,\rangle)$$

$$\frac{\langle x, \rho \rangle \xrightarrow{\mathsf{t}?m} \langle x', \rho' \rangle}{\langle x /_{\mathsf{t}} H, \rho \rangle {\uparrow}}\ H(\langle m \rangle) = \mathsf{B} \qquad \frac{\langle x, \rho \rangle \xrightarrow{f?m} \langle x', \rho' \rangle}{\langle x /_f H, \rho \rangle {\uparrow}}\ H(\langle m \rangle) = \mathsf{R}$$

$$\frac{\langle x, \rho \rangle \xrightarrow{f??m} \langle x', \rho' \rangle}{\langle x /_f H, \rho \rangle \xrightarrow{\mathsf{tau}} \langle x' /_f H, \rho' \rangle}\ H(\langle m \rangle) \in \{\mathsf{T}, \mathsf{F}, \mathsf{B}\},\ (f?m, \mathsf{T}) \in \rho(\langle\,\rangle)$$

$$\frac{\langle x, \rho \rangle \xrightarrow{f??m} \langle x', \rho' \rangle}{\langle x /_f H, \rho \rangle \xrightarrow{\mathsf{tau}} \langle x' /_f H, \rho' \rangle}\ H(\langle m \rangle) = \mathsf{R},\ (f?m, \mathsf{F}) \in \rho(\langle\,\rangle)$$

If $f = \mathsf{t}$, then f is the focus of the local service of the thread $x \unlhd f?m \unrhd y$. No other thread can raise a state of this service in which the processing of m is blocked. Hence, if the processing of m is blocked, it is blocked forever.

The additional transition rules for thread-service composition in the presence of guarding tests are given in Table 12.

6 Delays and Exception Handling

We go on to show how guarding tests can used to express postconditional composition with delay and postconditional composition with exception handling.

For postconditional composition with delay, we extend the set of basic actions $\mathcal{A}$ with the set $\{f!m \mid f \in \mathcal{F}, m \in \mathcal{M}\}$. Performing a basic action $f!m$ is like performing $f.m$, but in case processing of the command m is temporarily blocked, it is automatically delayed until the blockade is over.

Postconditional composition with delay is defined by the equation given in Table 13. The equation from this table guarantees that $f.m$ is only performed if

Table 13. Defining equation for postconditional composition with delay

$$x \unlhd f!m \unrhd y = (x \unlhd f.m \unrhd y) \unlhd f?m \unrhd (x \unlhd f!m \unrhd y)$$

Table 14. Defining equations for postconditional composition with exception handling

$$\begin{array}{l} x \unlhd f.m\,[y] \unrhd z = (x \unlhd f.m \unrhd z) \unlhd f??m \unrhd y \\ x \unlhd f!m\,[y] \unrhd z = ((x \unlhd f.m \unrhd z) \unlhd f?m \unrhd (x \unlhd f!m\,[y] \unrhd z)) \unlhd f??m \unrhd y \end{array}$$

$f?m$ yields a positive reply.

For postconditional composition with exception handling, we introduce the following notations: $x \unlhd f.m\,[y] \unrhd z$ and $x \unlhd f!m\,[y] \unrhd z$.

The intuition for $x \unlhd f.m\,[y] \unrhd z$ is that $x \unlhd f.m \unrhd z$ is tried, but y is done instead in the exceptional case that $x \unlhd f.m \unrhd z$ fails because the request to process m is refused. The intuition for $x \unlhd f!m\,[y] \unrhd z$ is that $x \unlhd f!m \unrhd z$ is tried, but y is done instead in the exceptional case that $x \unlhd f!m \unrhd z$ fails because the request to process m is refused. The processing of m may first be blocked and thereafter be refused; in that case, y is done instead as well.

The two forms of postconditional composition with exception handling are defined by the equations given in Table 14. The equations from this table guarantee that $f.m$ is only performed if $f??m$ yields a positive reply.

An alternative to the second equation from Table 14 is

$$x \unlhd f!m\,[y] \unrhd z = ((x \unlhd f.m \unrhd z) \unlhd f?m \unrhd (x \unlhd f!m \unrhd z)) \unlhd f??m \unrhd y\ .$$

In that case, y is only done if the processing of m is refused immediately.

7 A Formal Design Prototype

In this section, we show how the thread algebra developed so far can be used to give a simplified, formal representation schema of the design of systems that consist of several multi-threaded programs on various hosts in different networks. We propose to use the term *formal design prototype* for such a schema. The presented schema can be useful in understanding certain aspects of the system designed.

The set of *basic thread expressions*, with typical element P, is defined by

$$\begin{array}{rcl} P & ::= & \mathsf{D} \mid \mathsf{S} \mid P \unlhd f.m \unrhd P \mid P \unlhd f!m \unrhd P \mid \\ & & P \unlhd f.m\,[P] \unrhd P \mid P \unlhd f!m\,[P] \unrhd P \mid \langle X|E \rangle\ , \end{array}$$

where $f \in \mathcal{F}$, $m \in \mathcal{M}$ and $\langle X|E \rangle$ is a constant standing for the unique solution for variable X of a guarded recursive specification E in which the right-hand sides of the equations are basic thread expressions in which variables may occur wherever basic thread expressions are expected. Thus, the use of guarding tests, i.e. basic actions of the forms $f?m$ and $f??m$, is restricted to their intended use.

A thread with local service is described by an expression of the form $P \mathbin{/_{\mathsf{t}}} \mathit{TLS}$, where P is a basic thread expression and TLS is a local service for threads. TLS does nothing else but maintaining local data for a thread. A thread vector in which each thread has its local service is of the form

$$\langle P_1 \mathbin{/_{\mathsf{t}}} \mathit{TLS} \rangle \curvearrowright \ldots \curvearrowright \langle P_n \mathbin{/_{\mathsf{t}}} \mathit{TLS} \rangle \;,$$

where $P_1, \ldots, P_n$ are basic thread expressions.

A multi-thread with local service is described by an expression of the form $\|(\mathit{TV}) \mathbin{/_{\mathsf{p}}} \mathit{PLS}$, where TV is a thread vector in which each thread has its local service and PLS is a local service for multi-threads. PLS maintains shared data of the threads from which a multi-thread is composed. A typical example of such data are Java pipes. A multi-thread vector in which each multi-thread has its local service is of the form

$$\langle \|(\mathit{TV}_1) \mathbin{/_{\mathsf{p}}} \mathit{PLS} \rangle \curvearrowright \ldots \curvearrowright \langle \|(\mathit{TV}_m) \mathbin{/_{\mathsf{p}}} \mathit{PLS} \rangle \;,$$

where $\mathit{TV}_1, \ldots, \mathit{TV}_m$ are thread vectors in which each thread has its local service.

The behavior of a host with local service is described by an expression of the form $\|(\mathit{PV}) \mathbin{/_{\mathsf{h}}} \mathit{HLS}$, where PV is a multi-thread vector in which each multi-thread has its local service and HLS is a local service for hosts. HLS maintains shared data of the multi-threads on a host. A typical example of such data are the files connected with Unix sockets used for data transfer between multi-threads on the same host. A host behavior vector in which each host has its local service is of the form

$$\langle \|(\mathit{PV}_1) \mathbin{/_{\mathsf{h}}} \mathit{HLS} \rangle \curvearrowright \ldots \curvearrowright \langle \|(\mathit{PV}_l) \mathbin{/_{\mathsf{h}}} \mathit{HLS} \rangle \;,$$

where $\mathit{PV}_1, \ldots, \mathit{PV}_l$ are multi-thread vectors in which each multi-thread has its local service.

The behavior of a network with local service is described by an expression of the form $\|(\mathit{HV}) \mathbin{/_{\mathsf{n}}} \mathit{NLS}$, where HV is a host behavior vector in which each host has its local service and NLS is a local service for networks. NLS maintains shared data of the hosts in a network. A typical example of such data are the files connected with Unix sockets used for data transfer between different hosts in the same network. A network behavior vector in which each network has its local service is of the form

$$\langle \|(\mathit{HV}_1) \mathbin{/_{\mathsf{n}}} \mathit{NLS} \rangle \curvearrowright \ldots \curvearrowright \langle \|(\mathit{HV}_k) \mathbin{/_{\mathsf{n}}} \mathit{NLS} \rangle \;,$$

where $\mathit{HV}_1, \ldots, \mathit{HV}_k$ are host behavior vectors in which each host has its local service.

The behavior of a system that consist of several multi-threaded programs on various hosts in different networks is described by an expression of the form $\|(\mathit{NV})$, where NV is a network behavior vector in which each network has its local service.

Table 15. Definition of simulation relation

$\mathsf{S}\ \mathsf{sim}\ x$
$\mathsf{D}\ \mathsf{sim}\ x$
$x\ \mathsf{sim}\ y \wedge x\ \mathsf{sim}\ z \Rightarrow x\ \mathsf{sim}\ y \unlhd a \unrhd z$
$x\ \mathsf{sim}\ y \wedge z\ \mathsf{sim}\ w \Rightarrow x \unlhd a \unrhd z\ \mathsf{sim}\ y \unlhd a \unrhd w$

A typical example is the case where NV is an expression of the form

$$\begin{aligned}&\|(\langle\|(\langle\|(\langle P_1 /_{\mathsf{t}} \mathit{TLS}\rangle \curvearrowright \langle P_2 /_{\mathsf{t}} \mathit{TLS}\rangle) /_{\mathsf{p}} \mathit{PLS}\rangle \curvearrowright \\ &\qquad \langle\|(\langle P_3 /_{\mathsf{t}} \mathit{TLS}\rangle \curvearrowright \langle P_4 /_{\mathsf{t}} \mathit{TLS}\rangle \curvearrowright \langle P_5 /_{\mathsf{t}} \mathit{TLS}\rangle) /_{\mathsf{p}} \mathit{PLS}\rangle) /_{\mathsf{h}} \mathit{HLS}\rangle \curvearrowright \\ &\quad \langle\|(\langle\|(\langle P_6 /_{\mathsf{t}} \mathit{TLS}\rangle) /_{\mathsf{p}} \mathit{PLS}\rangle) /_{\mathsf{h}} \mathit{HLS}\rangle) /_{\mathsf{n}} \mathit{NLS}\ ,\end{aligned}$$

where $P_1, \ldots, P_6$ are basic thread expressions, and TLS, PLS, HLS and NLS are local services for threads, multi-threads, hosts and networks, respectively. It describes a system that consists of two hosts in one network, where on the first host currently a multi-thread with two threads and a multi-thread with three threads exist concurrently, and on the second host currently a single multi-thread with a single thread exists.

Evidence of correctness of the schema $\|(NV)$ is obtained by Lemma 1 given below. This lemma is phrased in terms of a simulation relation sim on the closed terms of the thread algebra developed in the preceding sections. The relation sim (is simulated by) is defined inductively by means of the rules in Table 15.

Lemma 1 (Simulation Lemma). *Let P be a basic thread expression in which all basic actions are from the set $\{f.m \mid f \in \mathcal{F} \setminus \{\mathsf{t}, \mathsf{p}, \mathsf{h}, \mathsf{n}\}, m \in \mathcal{M}\}$ and constants standing for the solutions of guarded recursive specifications do not occur. Let $C[P]$ be a context of P of the form $\|(NV)$ where NV is a network behavior vector as above. Then $P\ \mathsf{sim}\ C[P]$. This implies that $C[P]$ will perform all steps of P in finite time.*

Proof. First we prove $P\ \mathsf{sim}\ C'[P]$, where C' is a context of P of the form $\|(TV)$, by induction on the depth of P, and in both the basis and the inductive step, by induction on the position of P in thread vector TV. Using in each case the preceding result, we prove an analogous result for each higher-level vector in a similar way. □

8 Thread Identity Management in Local Services

A multi-thread with local service is described by an expression of the form $\|(TV) /_{\mathsf{p}} \mathit{PLS}$, where TV is a thread vector and PLS is a local service for multi-threads. When the local service PLS receives a request from the multi-thread $\|(TV)$, it often needs to know from which of the interleaved threads the request originates. This can be achieved by informing the local service whenever threads succeed each other by interleaving and whenever a thread drops out by

Table 16. Axioms for cyclic interleaving with thread identity management support

$\Vert_\ell(\langle\,\rangle) = \mathsf{S}$	CSItim1
$\Vert_\ell(\langle \mathsf{S} \rangle \frown \alpha) = \ell.\mathsf{shift} \circ \Vert_\ell(\alpha)$	CSItim2
$\Vert_\ell(\langle \mathsf{D} \rangle \frown \alpha) = \ell.\mathsf{shift} \circ \mathsf{S}_\mathsf{D}(\Vert_\ell(\alpha))$	CSItim3
$\Vert_\ell(\langle \mathsf{tau} \circ x \rangle \frown \alpha) = \mathsf{tau} \circ \ell.\mathsf{rotate} \circ \Vert_\ell(\alpha \frown \langle x \rangle)$	CSItim4
$\Vert_\ell(\langle x \unlhd f.m \unrhd y \rangle \frown \alpha) = \ell.\mathsf{rotate} \circ \Vert_\ell(\alpha \frown \langle x \rangle) \unlhd f.m \unrhd \ell.\mathsf{rotate} \circ \Vert_\ell(\alpha \frown \langle y \rangle)$	CSItim5
$\Vert_\ell(\langle x \unlhd f?m \unrhd y \rangle \frown \alpha) = \Vert_\ell(\langle x \rangle \frown \alpha) \unlhd f?m \unrhd \ell.\mathsf{rotate} \circ \Vert_\ell(\alpha \frown \langle y \rangle)$	CSItim6
$\Vert_\ell(\langle x \unlhd f??m \unrhd y \rangle \frown \alpha) = \Vert_\ell(\langle x \rangle \frown \alpha) \unlhd f??m \unrhd \ell.\mathsf{rotate} \circ \Vert_\ell(\alpha \frown \langle y \rangle)$	CSItim7

termination or a deadlock. Similar remarks apply to local services of hosts and networks.

That leads us to cyclic interleaving with thread identity management support. For this variation of cyclic interleaving, it is assumed that $\mathsf{rotate}, \mathsf{shift} \in \mathcal{M}$. Three new strategic interleaving operators are introduced: $\Vert_\mathsf{p}(\_)$, $\Vert_\mathsf{h}(\_)$ and $\Vert_\mathsf{n}(\_)$. The operator $\Vert_\mathsf{p}(\_)$ differs from $\Vert(\_)$ in that it generates a basic action $\mathsf{p.rotate}$ whenever threads succeed each other and it generates a basic action $\mathsf{p.shift}$ whenever a thread drops out. The operators $\Vert_\mathsf{h}(\_)$ and $\Vert_\mathsf{n}(\_)$ differ from $\Vert(\_)$ analogously.

The axioms for cyclic interleaving with thread identity management support are given in Table 16, where ℓ stands for an arbitrary focus from the set $\{\mathsf{p}, \mathsf{h}, \mathsf{n}\}$.

We refrain from giving the additional transition rules for $\Vert_\mathsf{p}(\_)$, $\Vert_\mathsf{h}(\_)$ and $\Vert_\mathsf{n}(\_)$. They are obvious variations of the transition rules for $\Vert(\_)$.

In order to cover local services in which thread identity management is needed, we have to adapt the formal design prototype given in Section 7. A multi-thread with local service is now described by an expression of the form $\Vert_\mathsf{p}(TV) /_\mathsf{p} PLS$, where TV is a thread vector in which each thread has its local service and PLS is a local service for multi-threads. The behavior of a host with local service is now described by an expression of the form $\Vert_\mathsf{h}(PV) /_\mathsf{h} HLS$, where PV is a multi-thread vector in which each multi-thread has its local service and HLS is a local service for hosts. The behavior of a network with local service is now described by an expression of the form $\Vert_\mathsf{n}(HV) /_\mathsf{n} NLS$, where HV is a host behavior vector in which each host has its local service and NLS is a local service for networks.

Notice that the forms of the expressions that describe a thread with local service and a system have not been adapted. In the first case, no interleaving of threads is involved; and in the second case, no local service is involved.

In Section 10, we will describe a service in which thread identity management is needed.

9 State-Based Description of Services

In this section, we introduce the state-based approach to describe services that will be used in Section 10 to describe a service in which thread identity management is needed. This approach is similar to the approach to describe state machines introduced in [14].

In this approach, a service is described by

- a set of states S;
- an initial state $s_0 \in S$;
- an effect function $\mathit{eff} : \mathcal{M} \times S \to S$;
- a yield function $\mathit{yld} : \mathcal{M} \times S \to \mathcal{R}$.

The set S contains the states in which the service may be; and the functions eff and yld give, for each method m and state s, the state and reply, respectively, that result from processing m in state s.

We define a cumulative effect function $\mathit{ceff} : \mathcal{M}^* \to S$ in terms of s_0 and eff as follows:

$$\begin{aligned}
&\mathit{ceff}(\langle\,\rangle) = s_0 \\
&\mathit{ceff}(\alpha \frown \langle m \rangle) = \mathit{eff}(m, \mathit{ceff}(\alpha))\ .
\end{aligned}$$

We define a service $H : \mathcal{M}^+ \to \mathcal{R}$ in terms of ceff and yld as follows:

$$H(\alpha \frown \langle m \rangle) = \mathit{yld}(m, \mathit{ceff}(\alpha))\ .$$

We consider H to be the service described by S, s_0, eff and yld.

Note that $H(\langle m \rangle) = \mathit{yld}(m, s_0)$ and $\frac{\partial}{\partial m} H$ is the service obtained by taking $\mathit{eff}(m, s_0)$ instead of s_0 as the initial state.

As an example, we give a state-based description of a very simple service concerning a Boolean cell. This service can be used as a local service of threads. It will be generalized in Section 10 to a service that can be used as a local service of multi-threads, hosts and networks.

It is assumed that $\mathcal{M}$ contains the following methods:

- $\mathsf{bc{:}set{:}T}$: the contents of the Boolean cell becomes T and the reply is T;
- $\mathsf{bc{:}set{:}F}$: the contents of the Boolean cell becomes F and the reply is F;
- $\mathsf{bc{:}get}$: nothing changes and the reply is the contents of the Boolean cell.

We write $\mathcal{M}_{\mathsf{bc}}$ for the set $\{\mathsf{bc{:}set{:}T}, \mathsf{bc{:}set{:}F}, \mathsf{bc{:}get}\}$.

The state-based description of the service is as follows:

- $S = \{\mathsf{T}, \mathsf{F}\}$;
- $s_0 = \mathsf{F}$;
- eff and yld are defined as follows:

$$\begin{array}{lll}
\mathit{eff}(\mathsf{bc{:}set{:}T}, s) = \mathsf{T}\ , & \mathit{yld}(\mathsf{bc{:}set{:}T}, s) = \mathsf{T}\ ; & \\
\mathit{eff}(\mathsf{bc{:}set{:}F}, s) = \mathsf{F}\ , & \mathit{yld}(\mathsf{bc{:}set{:}F}, s) = \mathsf{F}\ ; & \\
\mathit{eff}(\mathsf{bc{:}get}, s) = s\ , & \mathit{yld}(\mathsf{bc{:}get}, s) = s\ ; & \\
\mathit{eff}(m, s) = s\ , & \mathit{yld}(m, s) = \mathsf{R}\ , & \mathsf{if}\ m \notin \mathcal{M}_{\mathsf{bc}}\ .
\end{array}$$

In Section 12, we will show that services can also be viewed as processes that are definable over an extension of ACP with conditions introduced in [12].

10 Localizable Boolean Cells

In this section, we describe a service in which thread identity management is needed. It can be used as a local service of multi-threads, hosts and networks. The service, called LBC, concerns localizable Boolean cells. It generalizes the service described in Section 9. LBC is much simpler than a service maintaining Java pipes or a service maintaining the files connected with Unix sockets. However, its description suggests how to describe those more interesting services.

It is assumed that $\mathcal{M}$ contains all methods of LBC, to wit (for each $n \in \mathbb{N}$):

- $\mathsf{lbc}{:}n{:}\mathsf{create}$: if a Boolean cell with name n does not exist, it is created with status unowned and contents F, and the reply is T; otherwise, nothing changes and the reply is F;
- $\mathsf{lbc}{:}n{:}\mathsf{elim}$: if a Boolean cell with name n exists and it is unowned, it is eliminated and the reply is T; otherwise, nothing changes and the reply is F;
- $\mathsf{lbc}{:}n{:}\mathsf{claim}$: if a Boolean cell with name n exists and it is unowned or owned by the requesting thread, it becomes or remains owned by the requesting thread and the reply is T; otherwise, nothing changes and the reply is F if it does not exist and B if it is owned by a thread other than the requesting thread;
- $\mathsf{lbc}{:}n{:}\mathsf{release}$: if a Boolean cell with name n exists and it is owned by the requesting thread, it becomes unowned and the reply is T; otherwise, nothing changes and the reply is F if it does not exist and R if it is unowned or owned by a thread other than the requesting thread;
- $\mathsf{lbc}{:}n{:}\mathsf{set}{:}\mathsf{T}$: if a Boolean cell with name n exists and it is owned by the requesting thread, its contents becomes T and the reply is T; otherwise, nothing changes and the reply is R;
- $\mathsf{lbc}{:}n{:}\mathsf{set}{:}\mathsf{F}$: if a Boolean cell with name n exists and it is owned by the requesting thread, its contents becomes F and the reply is T; otherwise, nothing changes and the reply is R;
- $\mathsf{lbc}{:}n{:}\mathsf{get}$: if a Boolean cell with name n exists and it is owned by the requesting thread, nothing changes and the reply is its contents; otherwise, nothing changes as well and the reply is R.

We write $\mathcal{M}_{\mathsf{lbc}}$ for the set of all methods of LBC.

Notice that, formally, multi-threads and host behaviours are threads as well. Therefore, in the case where LBC is used as a local service of a host or a network, we can think of multi-thread or host where thread is written in the explanation of its methods given above.

We suppose that an instance of LBC knows, when it starts to service a multi-thread, host or network, the number of threads, multi-threads or hosts it has to deal with initially. We consider this number to be a parameter of the service.

Let $l_0 \in \mathbb{N}$. Then the state-based description of the service LBC with parameter l_0, written $LBC(l_0)$, is as follows:

$$S = \{(c, o, l) \in C \times O \times \mathbb{N} \mid \mathrm{dom}(c) = \mathrm{dom}(o), \max(\mathrm{rng}(o)) \leq l\} ,$$

where $C = \{c : N \to \{\mathsf{T}, \mathsf{F}\} \mid N \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N})\}$, $O = \{o : N \to \mathbb{N} \mid N \in \mathcal{P}_{\mathrm{fin}}(\mathbb{N})\}$; $s_0 = ([\,], [\,], l_0)$; and *eff* and *yld* are defined in Tables 17 and 18, respectively.

Table 17. Effect function for service with localizable Boolean cells

$\mathit{eff}(\mathsf{lbc}{:}n{:}\mathsf{create}, (c, o, l)) = (c \oplus [n \mapsto \mathsf{F}], o \oplus [n \mapsto 0], l)$	if $n \notin \mathrm{dom}(c)$
$\mathit{eff}(\mathsf{lbc}{:}n{:}\mathsf{create}, (c, o, l)) = (c, o, l)$	if $n \in \mathrm{dom}(c)$
$\mathit{eff}(\mathsf{lbc}{:}n{:}\mathsf{elim}, (c, o, l)) =$	
$\quad (c \restriction (\mathrm{dom}(c) \setminus \{n\}), o \restriction (\mathrm{dom}(c) \setminus \{n\}), l)$	if $n \in \mathrm{dom}(c) \ \land \ o(n) = 0$
$\mathit{eff}(\mathsf{lbc}{:}n{:}\mathsf{elim}, (c, o, l)) = (c, o, l)$	if $n \notin \mathrm{dom}(c) \ \lor \ o(n) \neq 0$
$\mathit{eff}(\mathsf{lbc}{:}n{:}\mathsf{claim}, (c, o, l)) = (c, o \oplus [n \mapsto 1], l)$	if $n \in \mathrm{dom}(c) \ \land \ o(n) \leq 1$
$\mathit{eff}(\mathsf{lbc}{:}n{:}\mathsf{claim}, (c, o, l)) = (c, o, l)$	if $n \notin \mathrm{dom}(c) \ \lor \ o(n) > 1$
$\mathit{eff}(\mathsf{lbc}{:}n{:}\mathsf{release}, (c, o, l)) = (c, o \oplus [n \mapsto 0], l)$	if $n \in \mathrm{dom}(c) \ \land \ o(n) = 1$
$\mathit{eff}(\mathsf{lbc}{:}n{:}\mathsf{release}, (c, o, l)) = (c, o, l)$	if $n \notin \mathrm{dom}(c) \ \lor \ o(n) \neq 1$
$\mathit{eff}(\mathsf{lbc}{:}n{:}\mathsf{set}{:}b, (c, o, l)) = (c \oplus [n \mapsto b], o, l)$	if $n \in \mathrm{dom}(c) \ \land \ o(n) = 1$
$\mathit{eff}(\mathsf{lbc}{:}n{:}\mathsf{set}{:}b, (c, o, l)) = (c, o, l)$	if $n \notin \mathrm{dom}(c) \ \lor \ o(n) \neq 1$
$\mathit{eff}(\mathsf{lbc}{:}n{:}\mathsf{get}, (c, o, l)) = (c, o, l)$	
$\mathit{eff}(\mathsf{rotate}, (c, o, l)) = (c, \mathit{rotate}(o, l), l)$	
$\mathit{eff}(\mathsf{shift}, (c, o, l)) = (c, \mathit{shift}(o, l), l - 1)$	
$\mathit{eff}(m, (c, o, l)) = (c, o, l)$	if $m \notin \mathcal{M}_{\mathsf{lbc}} \cup \{\mathsf{rotate}, \mathsf{shift}\}$

Table 18. Yield function for service with localizable Boolean cells

$\mathit{yld}(\mathsf{lbc}{:}n{:}\mathsf{create}, (c, o, l)) = \mathsf{T}$	if $n \notin \mathrm{dom}(c)$
$\mathit{yld}(\mathsf{lbc}{:}n{:}\mathsf{create}, (c, o, l)) = \mathsf{F}$	if $n \in \mathrm{dom}(c)$
$\mathit{yld}(\mathsf{lbc}{:}n{:}\mathsf{elim}, (c, o, l)) = \mathsf{T}$	if $n \in \mathrm{dom}(c) \ \land \ o(n) = 0$
$\mathit{yld}(\mathsf{lbc}{:}n{:}\mathsf{elim}, (c, o, l)) = \mathsf{F}$	if $n \notin \mathrm{dom}(c) \ \lor \ o(n) \neq 0$
$\mathit{yld}(\mathsf{lbc}{:}n{:}\mathsf{claim}, (c, o, l)) = \mathsf{T}$	if $n \in \mathrm{dom}(c) \ \land \ o(n) \leq 1$
$\mathit{yld}(\mathsf{lbc}{:}n{:}\mathsf{claim}, (c, o, l)) = \mathsf{F}$	if $n \notin \mathrm{dom}(c)$
$\mathit{yld}(\mathsf{lbc}{:}n{:}\mathsf{claim}, (c, o, l)) = \mathsf{B}$	if $n \in \mathrm{dom}(c) \ \land \ o(n) > 1$
$\mathit{yld}(\mathsf{lbc}{:}n{:}\mathsf{release}, (c, o, l)) = \mathsf{T}$	if $n \in \mathrm{dom}(c) \ \land \ o(n) = 1$
$\mathit{yld}(\mathsf{lbc}{:}n{:}\mathsf{release}, (c, o, l)) = \mathsf{F}$	if $n \notin \mathrm{dom}(c)$
$\mathit{yld}(\mathsf{lbc}{:}n{:}\mathsf{release}, (c, o, l)) = \mathsf{R}$	if $n \in \mathrm{dom}(c) \ \land \ o(n) \neq 1$
$\mathit{yld}(\mathsf{lbc}{:}n{:}\mathsf{set}{:}b, (c, o, l)) = \mathsf{T}$	if $n \in \mathrm{dom}(c) \ \land \ o(n) = 1$
$\mathit{yld}(\mathsf{lbc}{:}n{:}\mathsf{set}{:}b, (c, o, l)) = \mathsf{R}$	if $n \notin \mathrm{dom}(c) \ \lor \ o(n) \neq 1$
$\mathit{yld}(\mathsf{lbc}{:}n{:}\mathsf{get}, (c, o, l)) = c(n)$	if $n \in \mathrm{dom}(c) \ \land \ o(n) = 1$
$\mathit{yld}(\mathsf{lbc}{:}n{:}\mathsf{get}, (c, o, l)) = \mathsf{R}$	if $n \notin \mathrm{dom}(c) \ \lor \ o(n) \neq 1$
$\mathit{yld}(\mathsf{rotate}, (c, o, l)) = \mathsf{T}$	
$\mathit{yld}(\mathsf{shift}, (c, o, l)) = \mathsf{T}$	
$\mathit{yld}(m, (c, o, l)) = \mathsf{R}$	if $m \notin \mathcal{M}_{\mathsf{lbc}} \cup \{\mathsf{rotate}, \mathsf{shift}\}$

The state of the service comprises the contents (c) and owner (o) of the existing Boolean cells, and the number of threads, multi-threads or hosts it is dealing with (l). The functions $\mathit{rotate}, \mathit{shift} : O \times \mathbb{N} \to O$ used in Table 17 are defined as follows:

$$\mathrm{dom}(\mathit{rotate}(o,l)) = \mathrm{dom}(o)\ , \quad \mathrm{dom}(\mathit{shift}(o,l)) = \mathrm{dom}(o)\ ;$$

$$\begin{array}{lll} \mathit{rotate}(o,l)(n) = 0\ , & \mathit{shift}(o,l)(n) = 0\ , & \mathsf{if}\ o(n) = 0\ ; \\ \mathit{rotate}(o,l)(n) = l\ , & \mathit{shift}(o,l)(n) = 0\ , & \mathsf{if}\ o(n) = 1\ ; \\ \mathit{rotate}(o,l)(n) = o(n) - 1\ , & \mathit{shift}(o,l)(n) = o(n) - 1\ , & \mathsf{if}\ 1 < o(n) \leq l\ . \end{array}$$

We use the following notation for functions: $[\,]$ for the empty function; $[d \mapsto r]$ for the function f with $\mathrm{dom}(f) = \{d\}$ such that $f(d) = r$; $f \oplus g$ for the function h with $\mathrm{dom}(h) = \mathrm{dom}(f) \cup \mathrm{dom}(g)$ such that for all $d \in \mathrm{dom}(h)$, $h(d) = f(d)$ if $d \notin \mathrm{dom}(g)$ and $h(d) = g(d)$ otherwise; and $f \restriction D$ for the function g with $\mathrm{dom}(g) = \mathrm{dom}(f) \setminus D$ such that for all $d \in \mathrm{dom}(g)$, $g(d) = f(d)$.

11 ACP with Conditions

In Section 12, we will investigate the connections of threads and services with the processes considered in ACP-style process algebras. We will focus on ACP$^\mathrm{c}$, an extension of ACP with conditions introduced in [12]. In this section, we shortly review ACP$^\mathrm{c}$. For a comprehensive overview, the reader is referred to [12, 13]. The axioms of ACP$^\mathrm{c}$ are given in Appendix A.

ACP$^\mathrm{c}$ is an extension of ACP with conditional expressions in which the conditions are taken from a Boolean algebra. ACP$^\mathrm{c}$ has two sorts: (i) the sort $\mathbf{P}$ of *processes*, (ii) the sort $\mathbf{C}$ of *conditions*. In ACP$^\mathrm{c}$, it is assumed that the following has been given: a fixed but arbitrary set A (of actions), with $\delta \notin \mathsf{A}$, a fixed but arbitrary commutative and associative function $\mid : \mathsf{A} \cup \{\delta\} \times \mathsf{A} \cup \{\delta\} \to \mathsf{A} \cup \{\delta\}$ such that $\delta \mid a = \delta$ for all $a \in \mathsf{A} \cup \{\delta\}$, and a fixed but arbitrary set $\mathsf{C_{at}}$ (of atomic conditions). Henceforth, we write A_δ for $\mathsf{A} \cup \{\delta\}$.

Let p and q be closed terms of sort $\mathbf{P}$, ζ and ξ be closed term of sort $\mathbf{C}$, $a \in \mathsf{A}$, $H \subseteq \mathsf{A}$, and $\eta \in \mathsf{C_{at}}$. Intuitively, the constants and operators to build terms of sort $\mathbf{P}$ that will be used to define the processes to which threads and services correspond can be explained as follows:

- δ can neither perform an action nor terminate successfully;
- a first performs action a unconditionally and then terminates successfully;
- $p + q$ behaves either as p or as q, but not both;
- $p \cdot q$ first behaves as p, but when p terminates successfully it continues as q;
- $\zeta :\to p$ behaves as p under condition ζ;
- $p \parallel q$ behaves as the process that proceeds with p and q in parallel;
- $\partial_H(p)$ behaves the same as p, except that actions from H are blocked.

Intuitively, the constants and operators to build terms of sort $\mathbf{C}$ that will be used to define the processes to which threads and services correspond can be explained as follows:

- η is an atomic condition;
- $\bot$ is a condition that never holds;
- $\top$ is a condition that always holds;
- $-\zeta$ is the opposite of ζ;
- $\zeta \sqcup \xi$ is either ζ or ξ;
- $\zeta \sqcap \xi$ is both ζ and ξ.

The remaining operators of $\mathrm{ACP}^{\mathrm{c}}$ are of an auxiliary nature. They are needed to axiomatize $\mathrm{ACP}^{\mathrm{c}}$.

We write $\sum_{i \in \mathcal{I}} p_i$, where $\mathcal{I} = \{i_1, \ldots, i_n\}$ and $p_{i_1}, \ldots, p_{i_n}$ are terms of sort $\mathbf{P}$, for $p_{i_1} + \ldots + p_{i_n}$. The convention is that $\sum_{i \in \mathcal{I}} p_i$ stands for δ if $\mathcal{I} = \emptyset$. We use the notation $p \lhd \zeta \rhd q$, where p and q are terms of sort $\mathbf{P}$ and ζ is a term of sort $\mathbf{C}$, for $\zeta :\rightarrow p + -\zeta :\rightarrow q$.

A process is considered definable over $\mathrm{ACP}^{\mathrm{c}}$ if there exists a guarded recursive specification over $\mathrm{ACP}^{\mathrm{c}}$ that has that process as its solution.

A *recursive specification* over $\mathrm{ACP}^{\mathrm{c}}$ is a set of equations $E = \{X = t_X \mid X \in V\}$, where V is a set of variables and each t_X is a term of sort $\mathbf{P}$ that only contains variables from V. Let t be a term of sort $\mathbf{P}$ containing a variable X. An occurrence of X in t is *guarded* if t has a subterm of the form $a \cdot t'$ containing this occurrence of X. A recursive specification over $\mathrm{ACP}^{\mathrm{c}}$ is *guarded* if all occurrences of variables in the right-hand sides of its equations are guarded or it can be rewritten to such a recursive specification using the axioms of $\mathrm{ACP}^{\mathrm{c}}$ and the equations of the recursive specification. We only consider models of $\mathrm{ACP}^{\mathrm{c}}$ in which guarded recursive specifications have unique solutions.

For each guarded recursive specification E and each variable X that occurs as the left-hand side of an equation in E, we introduce a constant of sort $\mathbf{P}$ standing for the unique solution of E for X. This constant is denoted by $\langle X|E \rangle$. The axioms for guarded recursion are also given in Appendix A.

In order to express thread-service composition on the $\mathrm{ACP}^{\mathrm{c}}$-definable processes corresponding to threads and services, we need an extension of $\mathrm{ACP}^{\mathrm{c}}$ with renaming operators ρ_r like the ones introduced for ACP in [7]. Intuitively, the action renaming operator ρ_r, where $r : \mathsf{A} \rightarrow \mathsf{A}$, can be explained as follows: $\rho_r(p)$ behaves as p with each action replaced according to r. The axioms for action renaming are also given in Appendix A.

In order to explain the connection of threads and services with $\mathrm{ACP}^{\mathrm{c}}$ fully, we need an extension of $\mathrm{ACP}^{\mathrm{c}}$ with the condition evaluation operators CE_h introduced in [12]. Intuitively, the condition evaluation operator CE_h, where h is a function on conditions that is preserved by $\bot$, $\top$, $-$, $\sqcup$ and $\sqcap$, can be explained as follows: $\mathsf{CE}_h(p)$ behaves as p with each condition replaced according to h. The important point is that, if $h(\zeta) \in \{\bot, \top\}$, all subterms of the form $\zeta :\rightarrow q$ can be eliminated. The axioms for condition evaluation are also given in Appendix A.

12 Connections of Threads and Services with $\mathrm{ACP}^{\mathrm{c}}$

In this section, we show that threads and services can be viewed as processes that are definable over $\mathrm{ACP}^{\mathrm{c}}$, the extension of ACP with conditions reviewed

Table 19. Definition of translation function for threads

$$
\begin{aligned}
&\llbracket X \rrbracket = X \;, \\
&\llbracket \mathsf{S} \rrbracket = \mathrm{stop} \;, \\
&\llbracket \mathsf{D} \rrbracket = \mathrm{i} \cdot \delta \;, \\
&\llbracket t_1 \trianglelefteq \mathsf{tau} \trianglerighteq t_2 \rrbracket = \mathrm{i} \cdot \mathrm{i} \cdot \llbracket t_1 \rrbracket \;, \\
&\llbracket t_1 \trianglelefteq f.m \trianglerighteq t_2 \rrbracket = \mathrm{s}_f(m) \cdot (\mathrm{r}_f(\mathsf{T}) \cdot \llbracket t_1 \rrbracket + \mathrm{r}_f(\mathsf{F}) \cdot \llbracket t_2 \rrbracket) \;, \\
&\llbracket t_1 \trianglelefteq \mathsf{t}?m \trianglerighteq t_2 \rrbracket = \mathrm{s}_{\mathsf{t}}(?m) \cdot (\mathrm{r}_{\mathsf{t}}(\mathsf{T}) \cdot \llbracket t_1 \rrbracket + \mathrm{r}_{\mathsf{t}}(\mathsf{F}) \cdot \llbracket t_1 \rrbracket) \;, \\
&\llbracket t_1 \trianglelefteq f?m \trianglerighteq t_2 \rrbracket = \\
&\qquad \mathrm{s}_f(?m) \cdot (\mathrm{r}_f(\mathsf{T}) \cdot \llbracket t_1 \rrbracket + \mathrm{r}_f(\mathsf{F}) \cdot \llbracket t_1 \rrbracket + \mathrm{r}_f(\mathsf{B}) \cdot \llbracket t_2 \rrbracket) \qquad \text{if } f \neq \mathsf{t} \;, \\
&\llbracket t_1 \trianglelefteq f??m \trianglerighteq t_2 \rrbracket = \\
&\qquad \mathrm{s}_f(??m) \cdot (\mathrm{r}_f(\mathsf{T}) \cdot \llbracket t_1 \rrbracket + \mathrm{r}_f(\mathsf{F}) \cdot \llbracket t_1 \rrbracket + \mathrm{r}_f(\mathsf{B}) \cdot \llbracket t_1 \rrbracket + \mathrm{r}_f(\mathsf{R}) \cdot \llbracket t_2 \rrbracket) \;, \\
&\llbracket \langle X|E \rangle \rrbracket = \langle X | \{X = \llbracket t \rrbracket \mid X = t \in E\} \rangle \;.
\end{aligned}
$$

in Section 11, and that thread-service composition on those processes can be expressed in terms of operators of ACP$^{\mathrm{c}}$ with renaming.

For that purpose, A, $\mid$ and $\mathsf{C_{at}}$ are taken as follows:

$$
\begin{aligned}
\mathsf{A} \;=\; & \{\mathrm{s}_f(d) \mid f \in \mathcal{F}, d \in \mathcal{M} \cup \{?m \mid m \in \mathcal{M}\} \cup \{??m \mid m \in \mathcal{M}\} \cup \mathcal{R}\} \\
\cup\; & \{\mathrm{r}_f(d) \mid f \in \mathcal{F}, d \in \mathcal{M} \cup \{?m \mid m \in \mathcal{M}\} \cup \{??m \mid m \in \mathcal{M}\} \cup \mathcal{R}\} \\
\cup\; & \{\mathrm{stop}, \overline{\mathrm{stop}}, \mathrm{stop}^*, \mathrm{i}\} \;;
\end{aligned}
$$

for all $a \in \mathsf{A}$, $f \in \mathcal{F}$ and $d \in \mathcal{M} \cup \{?m \mid m \in \mathcal{M}\} \cup \{??m \mid m \in \mathcal{M}\} \cup \mathcal{R}$:

$$
\begin{array}{llll}
\mathrm{s}_f(d) \mid \mathrm{r}_f(d) = \mathrm{i} \;, & & \mathrm{stop} \mid \overline{\mathrm{stop}} = \mathrm{stop}^* \;, & \\
\mathrm{s}_f(d) \mid a = \delta & \text{if } a \neq \mathrm{r}_f(d) \;, & \mathrm{stop} \mid a = \delta & \text{if } a \neq \overline{\mathrm{stop}} \;, \\
a \mid \mathrm{r}_f(d) = \delta & \text{if } a \neq \mathrm{s}_f(d) \;, & a \mid \overline{\mathrm{stop}} = \delta & \text{if } a \neq \mathrm{stop} \;,
\end{array}
$$

$$
\mathrm{i} \mid a = \delta \;;
$$

and

$$
\mathsf{C_{at}} = \{H(\langle m \rangle) = r \mid H \in \mathcal{RF}, m \in \mathcal{M}, r \in \mathcal{R}\} \cup \{f = g \mid f, g \in \mathcal{F}\} \;.
$$

We proceed with relating threads and services to processes definable over ACP$^{\mathrm{c}}$. First of all, we define a function $\llbracket \_ \rrbracket$ that gives a translation of terms of the thread algebra developed in Sections 3–5 to terms of ACP$^{\mathrm{c}}$. The translation is restricted to the terms in which the operators for cyclic interleaving, deadlock at termination, and thread-service composition do not occur. It is easy to prove by induction that each terms of the thread algebra is derivably equal to a term in which these operators do not occur. Hence, the restriction does not cause any loss of generality. The function $\llbracket \_ \rrbracket$ is defined inductively by the equations given in Table 19. In Section 6, postconditional composition with delay and postconditional composition with exception handling are defined over the thread

Table 20. Definition of translation function for services

$[\![H]\!]_f = \langle P^f_H | E \rangle$

where E consists of an equation

$$\begin{aligned} P^f_{H'} = \sum_{m \in \mathcal{M}} & (\mathrm{r}_f(m) \cdot \mathrm{s}_f(H'(\langle m \rangle)) \cdot (P^f_{\frac{\partial}{\partial m} H'} \lhd H'(\langle m \rangle) = \mathsf{T} \sqcup H'(\langle m \rangle) = \mathsf{F} \rhd P^f_{H'}) \\ & + (\mathrm{r}_f(?m) + \mathrm{r}_f(??m)) \cdot \mathrm{s}_f(H'(\langle m \rangle)) \cdot P^f_{H'}) + \overline{\mathrm{stop}} \end{aligned}$$

for each $H' \in \mathcal{RF}$

algebra developed in Sections 3–5. Thus, the translation of a term of one of the additional forms ($t_1 \trianglelefteq f!m \trianglerighteq t_2$, $t_1 \trianglelefteq f.m\,[t_2] \trianglerighteq t_3$ or $t_1 \trianglelefteq f!m\,[t_2] \trianglerighteq t_3$) equals the translation of a term of the thread algebra developed in Sections 3–5:

$$\begin{aligned} & [\![t_1 \trianglelefteq f!m \trianglerighteq t_2]\!] = [\![\langle X | \{X = (t_1 \trianglelefteq f.m \trianglerighteq t_2) \trianglelefteq f?m \trianglerighteq X\} \rangle]\!] \;, \\ & [\![t_1 \trianglelefteq f.m\,[t_2] \trianglerighteq t_3]\!] = [\![(t_1 \trianglelefteq f.m \trianglerighteq t_3) \trianglelefteq f??m \trianglerighteq t_2]\!] \;, \\ & [\![t_1 \trianglelefteq f!m\,[t_2] \trianglerighteq t_3]\!] = \\ & \quad [\![\langle X | \{X = ((t_1 \trianglelefteq f.m \trianglerighteq t_3) \trianglelefteq f?m \trianglerighteq X) \trianglelefteq f??m \trianglerighteq t_2\} \rangle]\!] \;. \end{aligned}$$

Secondly, we define functions $[\![\_]\!]_f$, one for each $f \in \mathcal{F}$, that give translations of the services introduced in Section 4 to terms of ACP$^\mathrm{c}$. The translation of a service depends upon the focus associated with it. If focus f is associated with service H, it will only process basic actions that are of the form $f.m$. In that case, $[\![H]\!]_f$ is the correct translation. For every $f \in \mathcal{F}$, the function $[\![\_]\!]_f$ is defined in Table 20.

Notice that ACP is sufficient for the translation of threads: no conditional expressions occur in the translations. For the translation of services, we have used the full power of ACP$^\mathrm{c}$.

Next, we relate thread-service composition to operators of ACP$^\mathrm{c}$ with renaming. That is, we extend the translation function $[\![\_]\!]$ to terms in which thread-service composition does occur. The additional equation for this extension is given in Table 21.

The translations given above preserve the closed substitution instances of all axioms in which the operators for cyclic interleaving and deadlock at termination do not occur, i.e. axioms T1 and TSC1–TSC14 (see Tables 1, 7 and 11). Roughly speaking, this means that the translations of the closed substitution instances of these axioms are derivable from the axioms of ACP$^\mathrm{c}$. Axioms TSC1–TSC14 are for the greater part conditional equations. The conditions concerned take part in the translation as well. The conditions are looked upon as propositions with the conditions of the forms $H(\langle m \rangle) = r$ and $f = g$, i.e. the elements of $\mathsf{C_{at}}$, as propositional variables.

We define a function $[\![\_]\!]$ that gives a translation of conditional equations of the thread algebra developed in Sections 3–5 to equations of ACP$^\mathrm{c}$. For conve-

Table 21. Extension of translation function for threads to thread-service composition

$$[\![t \mathbin{/_f} H]\!] = \rho_r(\partial_{C_f}([\![t]\!] \parallel [\![H]\!]_f))$$

where r is such that

$$r(\mathrm{stop}^*) = \mathrm{stop} \qquad r(a) = a \text{ if } a \neq \mathrm{stop}^*$$

and C_f is defined by

$$\begin{aligned} C_f = {} & \{\mathrm{s}_f(d) \mid d \in \mathcal{M} \cup \{?m \mid m \in \mathcal{M}\} \cup \{??m \mid m \in \mathcal{M}\} \cup \mathcal{R}\} \\ & \cup \{\mathrm{r}_f(d) \mid d \in \mathcal{M} \cup \{?m \mid m \in \mathcal{M}\} \cup \{??m \mid m \in \mathcal{M}\} \cup \mathcal{R}\} \\ & \cup \{\mathrm{stop}, \overline{\mathrm{stop}}\} \end{aligned}$$

nience, unconditional equations are considered to be conditional equations with condition T. The function $[\![\_]\!]$ is defined as follows:

$$[\![t_1 = t_2 \text{ if } \phi]\!] \;=\; \mathsf{CE}_{h_{\Phi\cup\{\phi\}}}([\![p]\!]) = \mathsf{CE}_{h_{\Phi\cup\{\phi\}}}([\![q]\!])\ ,$$

where

$$\begin{aligned} \Phi = {} & \{\textstyle\bigwedge_{r\in\mathcal{R}} \neg\,(H(\langle m\rangle) = r \;\wedge\; \bigvee_{r'\in\mathcal{R}\setminus\{r\}} H(\langle m\rangle) = r') \mid H \in \mathcal{RF}, m \in \mathcal{M}\} \\ \cup\; & \{\textstyle\bigwedge_{f\in\mathcal{F}} f = f \;\wedge\; \bigwedge_{f\in\mathcal{F}} \bigwedge_{f'\in\mathcal{F}\setminus\{f\}} \neg\; f = f'\}\ . \end{aligned}$$

Here h_Ψ is a function on conditions of ACP$^{\mathrm{c}}$ that preserves $\bot$, $\top$, $-$, $\sqcup$ and $\sqcap$ and satisfies $h_\Psi(\alpha) = \top$ iff α corresponds to a proposition derivable from Ψ and $h_\Psi(\alpha) = \bot$ iff $-\alpha$ corresponds to a proposition derivable from Ψ.[5]

Theorem 1 (Preservation Theorem). *Let* $p = q$ if ϕ *be a closed substitution instance of* T1*,* TSC1*,* TSC2*, . . . ,* TCS13 *or* TSC14*. Then* $[\![p = q \text{ if } \phi]\!]$ *is derivable from* ACP$^{\mathrm{c}}$*.*

Proof. The proof is straightforward. We outline the proof for axiom TSC5. The other axioms are proved in a similar way. In the outline of the proof for axiom TSC5, E, r and C_f are as in Tables 20 and 21, and Φ is as above. We take an arbitrary closed substitution instance of TSC5, say

$$(p \trianglelefteq f.m \trianglerighteq q) \mathbin{/_f} H = \mathsf{tau} \circ (p \mathbin{/_f} \tfrac{\partial}{\partial m} H) \text{ if } H(\langle m\rangle) = \mathsf{T}\ .$$

The following equation about the translation of the left-hand side of the closed substitution instance of TSC5 is derivable from the axioms of ACP$^{\mathrm{c}}$ and the axioms for guarded recursive specifications over ACP$^{\mathrm{c}}$:

$$\begin{aligned} & \rho_r(\partial_{C_f}(\mathrm{s}_f(m) \cdot (\mathrm{r}_f(\mathsf{T}) \cdot [\![p]\!] + \mathrm{r}_f(\mathsf{F}) \cdot [\![q]\!]) \parallel \langle P^f_H | E\rangle)) \\ & \quad = \mathrm{i} \cdot \mathrm{i} \cdot (H(\langle m\rangle) = \mathsf{T} :\rightarrow \rho_r(\partial_{C_f}([\![p]\!] \parallel \langle P^f_{\frac{\partial}{\partial m}H} | E\rangle)) \\ & \qquad\qquad + H(\langle m\rangle) = \mathsf{F} :\rightarrow \rho_r(\partial_{C_f}([\![q]\!] \parallel \langle P^f_{\frac{\partial}{\partial m}H} | E\rangle)))\ . \end{aligned}$$

[5] Here we use "corresponds to" for the wordy "is isomorphic to the equivalence class with respect to logical equivalence of" (see also [12]).

The following equation is derivable from this equation and the axioms for condition evaluation:

$$\begin{aligned}
&\mathsf{CE}_{\Phi \cup \{H(\langle m \rangle) = \mathsf{T}\}}(\rho_r(\partial_{C_f}(\mathrm{s}_f(m) \cdot (\mathrm{r}_f(\mathsf{T}) \cdot \llbracket p \rrbracket + \mathrm{r}_f(\mathsf{F}) \cdot \llbracket q \rrbracket) \parallel \langle P^f_H | E \rangle))) \\
&\quad = \mathrm{i} \cdot \mathrm{i} \cdot \mathsf{CE}_{\Phi \cup \{H(\langle m \rangle) = \mathsf{T}\}}(\rho_r(\partial_{C_f}(\llbracket p \rrbracket \parallel \langle P^f_{\frac{\partial}{\partial m} H} | E \rangle))) \;.
\end{aligned}$$

The following equation about the translation of the right-hand side of the closed substitution instance of TSC5 is derivable from the axioms for condition evaluation:

$$\begin{aligned}
&\mathsf{CE}_{\Phi \cup \{H(\langle m \rangle) = \mathsf{T}\}}(\mathrm{i} \cdot \mathrm{i} \cdot \rho_r(\partial_{C_f}(\llbracket p \rrbracket \parallel \langle P^f_{\frac{\partial}{\partial m} H} | E \rangle))) \\
&\quad = \mathrm{i} \cdot \mathrm{i} \cdot \mathsf{CE}_{\Phi \cup \{H(\langle m \rangle) = \mathsf{T}\}}(\rho_r(\partial_{C_f}(\llbracket p \rrbracket \parallel \langle P^f_{\frac{\partial}{\partial m} H} | E \rangle))) \;.
\end{aligned}$$

Hence, the evaluated translation of the the left-hand side equals the evaluated translation of the the right-hand side. □

The statement that threads and services can be viewed as processes that are definable over ACP$^{\mathrm{c}}$ is justified by the fact that the translations given above preserve the closed substitution instances of all axioms concerned.

Suppose that we could also translate terms in which the operators for cyclic interleaving and deadlock at termination do occur such that the closed substitution instances of axioms CSI1–CSI7 and S2D1–S2D6 (see Tables 4 and 9) are preserved. This would give an even stronger justification. Moreover, the translation concerned would imply that we could apply the SRM-technique described in [4] to obtain a model of the thread algebra developed in Sections 3–5 from each minimal model of ACP$^{\mathrm{c}}$. The generalization of the SRM-technique described in [10], which is not restricted to minimal models, would make a first-order extension of ACP$^{\mathrm{c}}$ necessary.

However, we are not able to extend the translation function $\llbracket \_ \rrbracket$ to terms in which the operator for cyclic interleaving occurs. The operator for cyclic interleaving asks much more than the operator for thread-service composition. Basically, more advanced conditions than the conditions that can be expressed with the retrospection operator and the last action constants added to ACP$^{\mathrm{c}}$ in [12] should be added to ACP$^{\mathrm{c}}$. A sort of sequences of processes, with constants and operators belonging to it, should be added as well.

13 Conclusions

We have presented an algebraic theory of threads and multi-threads based on multi-level strategic interleaving for the simple strategy of cyclic interleaving. The other interleaving strategies treated in [11] can be adapted to the setting of multi-level strategic interleaving in a similar way. We have also presented a reasonable though simplified formal representation schema of the design of systems that consist of several multi-threaded programs on various hosts in different networks. By dealing with delays and exceptions, this schema is sufficiently expressive to formalize mechanisms like Java pipes (for communication between threads) and Unix sockets (for communication between multi-threads, called processes in Unix jargon, and communication between hosts). Such mechanisms calls for services in which thread identity management is needed. In the primary theory, multi-level strategic interleaving does not provide support of thread identity management by services. We have presented an adaptation of the primary theory that does provide support thereof. We have shown the connections of threads and services with processes that are definable over ACP$^{\mathrm{c}}$, an extension of ACP with conditions introduced in [12], as well.

To the best of our knowledge, there is no other work on the theory of threads and multi-threads that is based on strategic interleaving. Although a deterministic interleaving strategy is always used for thread interleaving, it is the practice in work in which the semantics of multi-threated programs is involved to look upon thread interleaving as arbitrary interleaving, see e.g. [1, 16].

One of the options for future work is to formalize mechanisms like Java pipes and Unix sockets using the thread algebra developed in this paper. Another option for future work is to adapt some interleaving strategies from [11], other than cyclic interleaving, to the setting of multi-level strategic interleaving.

A Axioms of ACP$^{\mathrm{c}}$

The axioms of ACP$^{\mathrm{c}}$ are given in Tables 22–23. The axioms for guarded recursive specifications over ACP$^{\mathrm{c}}$ are given in Table 24. The additional axioms for condition evaluation and action renaming are given in Tables 25 and 26, respectively. In Table 24, we use the following notation. Let E be a recursive specification over ACP$^{\mathrm{c}}$, and let t be a term of ACP$^{\mathrm{c}}$. Then we write $\mathrm{V}(E)$ for the set of all variables that occur on the left-hand side of an equation in E, and we write $\langle t|E\rangle$ for t with, for all $X \in \mathrm{V}(E)$, all occurrences of X in t replaced by $\langle X|E\rangle$.

Table 22. Axioms of BPA$^{\mathrm{c}}$

$x + y = y + x$	A1	$\top :\rightarrow x = x$	GC1
$(x + y) + z = x + (y + z)$	A2	$\bot :\rightarrow x = \delta$	GC2
$x + x = x$	A3	$\phi :\rightarrow \delta = \delta$	GC3
$(x + y) \cdot z = x \cdot z + y \cdot z$	A4	$\phi :\rightarrow (x + y) = \phi :\rightarrow x + \phi :\rightarrow y$	GC4
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	A5	$\phi :\rightarrow x \cdot y = (\phi :\rightarrow x) \cdot y$	GC5
$x + \delta = x$	A6	$\phi :\rightarrow (\psi :\rightarrow x) = (\phi \sqcap \psi) :\rightarrow x$	GC6
$\delta \cdot x = \delta$	A7	$(\phi \sqcup \psi) :\rightarrow x = \phi :\rightarrow x + \psi :\rightarrow x$	GC7
$\phi \sqcup \bot = \phi$	BA1	$\phi \sqcap \top = \phi$	BA5
$\phi \sqcup -\phi = \top$	BA2	$\phi \sqcap -\phi = \bot$	BA6
$\phi \sqcup \psi = \psi \sqcup \phi$	BA3	$\phi \sqcap \psi = \psi \sqcap \phi$	BA7
$\phi \sqcup (\psi \sqcap \chi) = (\phi \sqcup \psi) \sqcap (\phi \sqcup \chi)$	BA4	$\phi \sqcap (\psi \sqcup \chi) = (\phi \sqcap \psi) \sqcup (\phi \sqcap \chi)$	BA8

Table 23. Additional axioms for ACP$^{\mathrm{c}}$ $(a, b, c \in \mathsf{A}_\delta)$

$x \parallel y = x \mathbin{\lfloor\!\lfloor} y + y \mathbin{\lfloor\!\lfloor} x + x \mid y$	CM1	$\partial_H(a) = a$	if $a \notin H$	D1
$a \mathbin{\lfloor\!\lfloor} x = a \cdot x$	CM2	$\partial_H(a) = \delta$	if $a \in H$	D2
$a \cdot x \mathbin{\lfloor\!\lfloor} y = a \cdot (x \parallel y)$	CM3	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$		D3
$(x + y) \mathbin{\lfloor\!\lfloor} z = x \mathbin{\lfloor\!\lfloor} z + y \mathbin{\lfloor\!\lfloor} z$	CM4	$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$		D4
$a \cdot x \mid b = (a \mid b) \cdot x$	CM5			
$a \mid b \cdot x = (a \mid b) \cdot x$	CM6	$(\phi :\rightarrow x) \mathbin{\lfloor\!\lfloor} y = \phi :\rightarrow (x \mathbin{\lfloor\!\lfloor} y)$		GC8
$a \cdot x \mid b \cdot y = (a \mid b) \cdot (x \parallel y)$	CM7	$(\phi :\rightarrow x) \mid y = \phi :\rightarrow (x \mid y)$		GC9
$(x + y) \mid z = x \mid z + y \mid z$	CM8	$x \mid (\phi :\rightarrow y) = \phi :\rightarrow (x \mid y)$		GC10
$x \mid (y + z) = x \mid y + x \mid z$	CM9	$\partial_H(\phi :\rightarrow x) = \phi :\rightarrow \partial_H(x)$		GC11
$a \mid b = b \mid a$	C1			
$(a \mid b) \mid c = a \mid (b \mid c)$	C2			
$\delta \mid a = \delta$	C3			

Table 24. Axioms for 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

Table 25. Axioms for condition evaluation $(a \in \mathsf{A}_\delta,\ \eta \in \mathsf{C}_{\mathsf{at}},\ \eta' \in \mathsf{C}_{\mathsf{at}} \cup \{\bot, \top\})$

$\mathsf{CE}_h(a) = a$	CE1	$\mathsf{CE}_h(\bot) = \bot$		CE6
$\mathsf{CE}_h(a \cdot x) = a \cdot \mathsf{CE}_h(x)$	CE2	$\mathsf{CE}_h(\top) = \top$		CE7
$\mathsf{CE}_h(x + y) = \mathsf{CE}_h(x) + \mathsf{CE}_h(y)$	CE3	$\mathsf{CE}_h(\eta) = \eta'$	if $h(\eta) = \eta'$	CE8
$\mathsf{CE}_h(\phi :\rightarrow x) = \mathsf{CE}_h(\phi) :\rightarrow \mathsf{CE}_h(x)$	CE4	$\mathsf{CE}_h(-\phi) = -\mathsf{CE}_h(\phi)$		CE9
$\mathsf{CE}_h(\mathsf{CE}_{h'}(x)) = \mathsf{CE}_{h \circ h'}(x)$	CE5	$\mathsf{CE}_h(\phi \sqcup \psi) = \mathsf{CE}_h(\phi) \sqcup \mathsf{CE}_h(\psi)$		CE10
		$\mathsf{CE}_h(\phi \sqcap \psi) = \mathsf{CE}_h(\phi) \sqcap \mathsf{CE}_h(\psi)$		CE11

Table 26. Axioms for action renaming $(a \in \mathsf{A})$

$\rho_r(\delta) = \delta$	ARN1
$\rho_r(a) = r(a)$	ARN2
$\rho_r(a \cdot x) = r(a) \cdot \rho_r(x)$	ARN3
$\rho_r(x + y) = \rho_r(x) + \rho_r(y)$	ARN4
$\rho_r(\phi :\rightarrow x) = \phi :\rightarrow \rho_r(x)$	ARN5

References

1. E. Ábrahám, F. S. de Boer, W. P. de Roever, and M. Steffen. A compositional operational semantics for JavaMT. In N. Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *Lecture Notes in Computer Science*, pages 290–303. Springer-Verlag, 2003.
2. L. Aceto, W. J. Fokkink, and C. Verhoef. Structural operational semantics. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, pages 197–292. Elsevier, Amsterdam, 2001.
3. K. Arnold and J. Gosling. *The Java Programming Language*. Addison-Wesley, 1996.
4. J. C. M. Baeten and J. A. Bergstra. On sequential composition, action prefixes and process prefix. *Formal Aspects of Computing*, 6:250–268, 1994.
5. J. A. Bergstra and I. Bethke. Polarized process algebra and program equivalence. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Proceedings 30th ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 1–21. Springer-Verlag, 2003.
6. J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1/3):109–137, 1984.
7. 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, *Proceedings Mathematics and Computer Science II*, volume 4 of *CWI Monograph*, pages 61–94. North-Holland, 1986.
8. J. A. Bergstra and M. E. Loots. Program algebra for sequential code. *Journal of Logic and Algebraic Programming*, 51(2):125–156, 2002.
9. J. A. Bergstra and C. A. Middelburg. A thread algebra with multi-level strategic interleaving. To appear in B. Cooper, B. Loewe and L. Torenvliet, editors, *CiE 2005*, LNCS, Springer-Verlag, 2005. Preliminary version: Computing Science Report 04-41, Department of Mathematics and Computing Science, Eindhoven University of Technology.
10. J. A. Bergstra and C. A. Middelburg. Model theory for process algebra. Computer Science Report 04-24, Department of Mathematics and Computer Science, Eindhoven University of Technology, September 2004.
11. J. A. Bergstra and C. A. Middelburg. Thread algebra for strategic interleaving. Computer Science Report 04-35, Department of Mathematics and Computer Science, Eindhoven University of Technology, November 2004.
12. J. A. Bergstra and C. A. Middelburg. Splitting bisimulations and retrospective conditions. Computer Science Report 05-03, Department of Mathematics and Computer Science, Eindhoven University of Technology, January 2005.
13. J. A. Bergstra and C. A. Middelburg. Strong splitting bisimulation equivalence. Computer Science Report 05-04, Department of Mathematics and Computer Science, Eindhoven University of Technology, February 2005.
14. J. A. Bergstra and A. Ponse. Combining programs and state machines. *Journal of Logic and Algebraic Programming*, 51(2):175–192, 2002.
15. J. Bishop and N. Horspool. *C# Concisely*. Addison-Wesley, 2004.
16. C. Flanagan, S. N. Freund, S. Qadeer, and S. A. Seshia. Modular verification of multithreaded programs. To appear in *Theoretical Computer Science*, 2005.
17. C. A. Middelburg. An alternative formulation of operational conservativity with binding terms. *Journal of Logic and Algebraic Programming*, 55(1/2):1–19, 2003.