

Thread algebra for SANE Virtual Processors ^{*}

Thuy Duong Vu¹ and Chris Jesshope²
{tdvu,jesshope}@science.uva.nl

Sectie Software Engineering¹,
Computer Systems Architecture Group²,
University of Amsterdam,
The Netherlands

Abstract. This paper presents a formal approach to the verification and evaluation of a programming/machine model being developed at the University of Amsterdam, called the *SANE Virtual Processor* (SVP). The model is being used as a basis for designing and programming chip multiprocessors and to support self-adaptive computation. This model can provide solutions for effectively programming distributed multiprocessor systems [12, 13, 17]. We take thread algebra [11], a semantics for recent object-oriented programming languages such as C# and Java, as a theoretical framework of our approach. We model the basic interleaving strategies used in SVP, and show in thread algebra that the SVP programs has a desired property, i.e. they are communication-deadlock free.

1 Introduction

The microthreaded model has evolved over the last decade from a low-level machine model [12] to a high-level programming model that supports dynamic concurrency and communication in self-adaptive systems. The latter is captured as an abstract definition of a SANE Virtual Processor (SVP) and a C-based language that realizes this [15]. SANE refers to a *self adaptive network entity* and the SVP model was developed in the AETHER project (<http://www.aether-ist.org/>), the goal of which is to explore self-adaptivity in complex computing systems, which requires dynamic concurrency. The SVP model is based on a substantial amount of prior research that started with the DRISC microprocessor design [12] and developed through various concurrency models based on microthreading[18, 16, 13, 17]. SVP however, places concurrency at the core of the machine and operating system model and provides a full range of concurrency control mechanisms. There are controls to dynamically create concurrency as indexed families of threads, to provide instruction-level synchronization of data to manage sequence within a family and bulk synchronization, to provide efficient synchronization between the units of work defined by concurrent families

^{*} The work presented in this paper is supported by NWO (Netherlands Organisation for Scientific Research) in the "Foundations for Massively Parallel on-chip Architectures using Microthreading" project.

of threads. Finally there are controls to provide reflection on concurrency in terms of killing or preempting concurrent programs. It should be emphasized that composition in the model is concurrent and self-similar down to the lowest levels of threads in a program, i.e. threads within a family may create subordinate families.

The pragmatic goals of this work are twofold. The first is to have a model that can be implemented efficiently in silicon to provide scalable performance in the emerging era of chip multiprocessors. The second is to have a model that has all of the characteristics of the sequential model, namely determinism and freedom from deadlock but which exposes as much concurrency as possible from a program without implying any scheduling. The aim is to develop a model that can be used as a compiler target from sequential programs but which also embeds the system infrastructure of job creation and termination. This means that the model is capable of representing all concurrency from the instruction-level up to operating system functionality in a unified manner.

This means programs (or microthreads) must be free of deadlock by design and allow safe composition, so that given two SVP programs, those programs can be composed in a third SVP program and the resulting program will be well behaved, i.e. deadlock free and deterministic if two programs were composed deterministically. Here determinism means that program should always give the same result, a key property of the sequential paradigm.

This paper represents the first results from a collaboration between the Computer Systems Architecture group and Sectie Software Engineering at the University of Amsterdam. It provides formalization to the model SVP. It like the preceding work presented in [10, 7–9] is a part of a project investigating microthreading. We take *thread algebra* (TA) [6, 11] as a theoretical framework of our approach. Thread algebra is an algebraic theory for sequential and multi-threaded programming languages. It is designed on top of *basic thread algebra* (BTA) [5] and contains a collection of *strategic interleaving operators* for parallel threads capturing some essential aspects of multi-threading. Suppose that a single thread is defined in BTA. The strategic interleaving operators will turn a sequence of single threads to a thread in BTA as well. It has been outlined in [6, 11] that thread algebra is a dominant form for recent object-oriented programming languages such as C# and Java where arbitrary interleaving is not an appropriate intuition.

We will analyze SVP program behaviors (or threads) and their compositions (multi-threads) mathematically in the setting of thread algebra. We model two deterministic interleaving strategies that are used in SVP. The first and basic strategy in SVP is called the *current thread persistence* with *blocking* which executes the threads in a round-robin fashion. This strategy is a variant of the *current thread persistence* strategy in [11]. The second one deals with *synchronous cooperation* of threads which intend to perform simultaneously all independent instructions from the threads at every step in the execution for speeding up processors [24]. We will show that SVP programs have one of the desired properties, i.e. they are deadlock free.

The structure of this paper as follows. Section 2 recalls the concepts of BTA to define BTA_{svp} (basic thread algebra for SVP). Section 3 extends BTA_{svp} with the basic interleaving strategy used in SVP to TA_{svp} . Section 4 extends TA_{svp} with the strategy dealing with synchronous cooperation of microthreads in SVP. Section 5 presents a SOS for TA_{svp} and shows that SVP programs are free of communication-deadlock. Furthermore, the axioms of TA_{svp} are sound and complete with respect to bisimulation equivalence induced by this SOS. The paper is ended with some concluding remarks in Section 6.

2 Basic thread algebra for SANE virtual processors

Basic thread algebra (BTA) [11] was first introduced as *basic polarized process algebra* (BPPA) in [5], a semantics for sequential programming languages. This section recalls the basic concepts of BTA from [5, 11] to provide a formal semantics for SVP sequential programs, denoted by BTA_{svp} (*basic thread algebra for SVP*). The behavior of an SVP sequential program is considered as a *thread* in BTA_{svp} . Instructions of SVP programs are now called *actions*.

2.1 Finite threads in BTA_{svp}

We assume the existence of a fixed but arbitrary set \mathcal{BA} of *basic actions* in BTA_{svp} . Each basic action $a \in \mathcal{BA}$ of a thread is taken as a command to the execution environment of the thread. This command is accepted or rejected depending on a boolean value $?a$ produced by the execution environment. If $?a = T$ (**true**) then the action a is *independent* and can be executed otherwise it is *blocked*. The execution environment cannot do anything with it. If a is executed then at completion of the processing of a , the execution environment produces a reply value y_a . This reply is either T or F (**false**) and is returned to the thread concerned.

BTA_{svp} (basic thread algebra for SVP) is the extension of BTA with the *independence* and *reply* conditional operators [9]. Hence, the set A of finite threads in BTA_{svp} is defined inductively the following operators:

- *Successful termination*: $S \in A$ yields successful terminating behavior.
- *Unsuccessful termination* or *deadlock*: $D \in A$ represents inactive behavior.
- *Postconditional composition*: $- \triangleleft a \triangleright -$ with $a \in \mathcal{BA}$. The thread $p \triangleleft a \triangleright q$, where $p, q \in A$, first performs a and then proceeds with p if T was returned and with q otherwise. In case $p = q$ we abbreviate this thread by the *action prefix* operator: $a \circ -$. In particular, $a \circ p = p \triangleleft a \triangleright p$.
- The *independence* conditional operator: $- \triangleleft ?a \triangleright -$ with $a \in \mathcal{BA}$. The thread $p \triangleleft ?a \triangleright q$ behaves as p if a is independent ($?a = T$) and it behaves as q otherwise.
- The *reply* conditional operator: $- \triangleleft y_a \triangleright -$ with $a \in \mathcal{BA}$. Similarly, the thread $p \triangleleft y_a \triangleright q$ behaves as p if the execution of a return a positive reply T , and it behaves as q if the execution of a return a negative reply F .

$x \triangleleft T \triangleright y = x$	CO1
$x \triangleleft F \triangleright y = y$	CO2
$x \triangleleft c \triangleright x = x$	CO3
$(x \triangleleft c \triangleright y) \triangleleft c \triangleright z = x \triangleleft c \triangleright z$	CO4
$x \triangleleft c \triangleright (y \triangleleft c \triangleright z) = x \triangleleft c \triangleright z$	CO5
$(x \triangleleft c_1 \triangleright y) \triangleleft c_2 \triangleright z = (x \triangleleft c_2 \triangleright z) \triangleleft c_1 \triangleright (y \triangleleft c_2 \triangleright z)$	CO6
$x \triangleleft c_1 \triangleright (y \triangleleft c_2 \triangleright z) = (x \triangleleft c_1 \triangleright y) \triangleleft c_2 \triangleright (x \triangleleft c_1 \triangleright z)$	CO7
$x \trianglelefteq a \trianglerighteq y = a \circ (x \triangleleft y_a \triangleright y)$	RC1
$x \triangleleft \mathbf{y}\mathbf{tau} \triangleright y = x$	RC2
$x \triangleleft ?\mathbf{tau} \triangleright y = x$	IC

Table 1. Axioms for conditions.

$$\overline{x \triangleleft \mathbf{tau} \trianglerighteq y = \mathbf{tau} \circ x}$$

Table 2. Axioms for the concrete internal action.

We note that S and D are similar to the termination ϵ and the deadlock δ used in other process algebras such as CCS [20] and ACP [4]. Furthermore, the independence conditional operator and the reply conditional operator originate from the *conditional* operator [2] defined for process algebra, where the second argument must be T or F . Table 1 represents the axioms on conditions.

For a thread $p = p_1 \triangleleft c \triangleright p_2$, c is called a *guard* of p . If c' is a guard of p_1 or p_2 then it is also a guard of p . The threads p_1 and p_2 are called *residuals* of p . If p' is a residual of p_1 or p_2 then it is also a residual of p . A residual of p is a *real* one if it does not have a guard.

It should also be noted that in [11], when dealing with blocking of concurrent threads in *thread algebra*, Bergstra and Middelburg introduce guards of the form $?a$ as actions to request the execution of a to the execution environment. The *strategic interleaving operators* of thread algebra, that turn a sequence of threads to a single thread in BTA, may introduce guards into threads. This mechanism deals with blocking in general. However, the thread obtained (also called a *multithread*) may become complicated, and hard to analyze. We decide to keep the formalization of the model SVP as simple as possible. Thus, in this paper, we do not consider guards $?a$ as actions, and instead, we treat them as boolean values produced by the execution environment as explained above.

The *concrete internal action* $\mathbf{tau} \in \mathcal{BA}$ given in [6] plays a special role. Its execution will never change any state and always produces a positive reply. The axiom for this action is given in Table 2.

2.2 Approximation Induction Principle

Threads can be *infinite*. An infinite thread in BTA_{svp} is represented by a *projective sequence* consisting of its finite approximations. These finite approximations are defined inductively by means of the approximation operators $\pi_n(-)$ of depth n of threads with $n \in \mathbb{N}$ whose axioms on finite threads are given as P0-P5 in

$\pi_0(x)=D$	P0
$\pi_{n+1}(S)=S$	P1
$\pi_{n+1}(D)=D$	P2
$\pi_{n+1}(x \trianglelefteq a \trianglerighteq y)=\pi_n(x) \trianglelefteq a \trianglerighteq \pi_n(y)$	P3
$\pi_{n+1}(x \triangleleft y_a \triangleright y)=\pi_{n+1}(x) \triangleleft y_a \triangleright \pi_{n+1}(y)$	P4
$\pi_{n+1}(x \triangleleft ?a \triangleright y)=\pi_{n+1}(x) \triangleleft ?a \triangleright \pi_{n+1}(y)$	P5
<u>If $\pi_n(x) = \pi_n(y)$ for all $n \in \mathbb{N}$ then $x = y$</u>	<u>AIP</u>

Table 3. Axioms for approximation operators and induction principle.

Table 3. Note that axioms P4 and P5 makes use of the assumption that \mathcal{BA} is finite.

A *projective sequence* is a sequence $(p_n)_{n \in \mathbb{N}}$ such that for all $n \in \mathbb{N}$, $\pi_n(p_{n+1}) = p_n$.

The *Approximation Induction Principle* (AIP) in Table 3 states that two threads are considered identical if their finite approximations at every depth are identical. We write A^∞ for the set of (finite and infinite) threads, and $\pi_n(p)$ for the projection at depth n of a thread $p \in A^\infty$. A^∞ is called a *projective limit model*.

3 Thread algebra for SANE Virtual Processors

In this section, we extend BTA_{svp} with the basic interleaving strategy that is used in SVP to TA_{svp} (*thread algebra for SVP*). We call this strategy *current thread persistence with blocking* which works similarly to the *current thread persistence* strategy given in [11].

3.1 Thread creation

First of all, we will explain how a family of threads in SVP is created. Thread forking or thread creation has been studied in [11, 9] with imperfect forking (forking off a thread may be blocked and/or fail) and perfect forking. All of them deal with the creation of only one thread. In SVP, we deal with the forking of a *family* (or a sequence) of threads. Furthermore, we assume that there is no resource deadlock in thread creation. Hence thread creation considered in this paper is a perfect forking as the one described in [9].

Let $\langle \rangle$ denote the empty sequence, $\langle p \rangle$ the sequence having p as sole element, and $\alpha \curvearrowright \beta$ the concatenation of finite sequences α and β .

The creation of a family of threads in TA_{svp} is given by the *forking postconditional composition* operator $- \trianglelefteq \text{NT}(\alpha) \trianglerighteq -$ where α is a sequence of threads. The thread $p \trianglelefteq \text{NT}(\alpha) \trianglerighteq q$ for some threads p, q is called the *creating* thread of α . $\text{NT}(\alpha)$ is considered as a thread forking action. Like a real action, its execution also produces a reply. Since we only deal with perfect forking in this paper, this reply is always T .

$$\frac{\pi_{n+1}(x \trianglelefteq \text{NT}(\langle z_1 \rangle \curvearrowright \dots \curvearrowright \langle z_k \rangle) \triangleright y) = \pi_n(x \trianglelefteq \text{NT}(\langle \pi_n(z_1) \rangle \curvearrowright \dots \curvearrowright \langle \pi_n(z_k) \rangle) \triangleright \pi_n(y))}{\text{PNT}}$$

Table 4. Axioms for approximation operators with thread creation.

Let $\alpha = \langle p_1 \rangle \curvearrowright \dots \curvearrowright \langle p_n \rangle$. We say that p_i for $1 \leq i \leq n$ are the threads in the same family. Moreover, p_i is a *predecessor* of p_{i+1} for all $1 \leq i < n$, and p_n is a predecessor of the creating thread. Furthermore, if $r \neq p_n$ is a predecessor of the creating thread then r is also a predecessor of p_1 . In concurrency of SVP, the blocking of a thread in a sequence of concurrent threads is allowed in a very restricted manner, depending only on its predecessors. In other words, a thread may only be waiting for some data produced by its predecessors. Hence, dependencies between threads in SVP can be represented as an acyclic graph, which in turn ensures freedom from *communication-deadlock* in the model SVP (see Section 5). We note that communication-deadlock occurs if all concurrent threads are blocked by its current actions.

In Table 4 we present an axiom for the approximation operator for thread creation in TA_{svp} . This axiom is concise with the axiom for thread creation given in [11, 9] in the case that the sequence of threads to be created is of length one.

3.2 The current thread persistence with blocking

The axioms for current thread persistence with blocking are given in Table 5. We assume the existence of a special action $\mathbf{swch} \in \mathcal{BA}$ to switch off the current thread to another thread in the sequence of concurrent threads. This switching off may speed up processors in some cases. The composition of an empty sequence of threads will terminate successfully. If the first thread of the sequence is terminated then the execution proceeds with the subsequent threads. If the first thread is in deadlock then whole system is in deadlock. In the remaining case, the system will execute the actions of the first thread until there is a blocked action, or the action \mathbf{swch} . The control flow then proceeds with the next thread in the sequence. The first thread meanwhile is put to the end of the sequence in a round-robin fashion. When creating a new family of threads or switching off to another thread, the action \mathbf{tau} will arise as a residue to keep pace with other threads in the sequence. We note that the threads are supposed initially not to contain any guards.

The concurrency of a sequence containing only one thread, in the case that there is no thread forking, is the thread itself. Furthermore, if there is no action \mathbf{swch} in the threads, these threads are eventually executed sequentially. It is because the order of the threads in the execution is changed only in the case that a thread is blocked. However, this will never happen since its predecessors are either terminated or deadlock.

In [11], a similar strategic interleaving operator that deals with blocking of threads is also given, namely $\|_{ba}$. In this strategy, $?a$ is considered as an action as mentioned earlier. Threads in left-hand side of the axioms of $\|_{ba}$, therefore,

$\ _{ctpb} (\langle \rangle) = S$	Ctpb1
$\ _{ctpb} (\langle S \rangle \curvearrowright \alpha) = \ _{ctpb} (\alpha)$	Ctpb2
$\ _{ctpb} (\langle D \rangle \curvearrowright \alpha) = D$	Ctpb3
$\ _{ctpb} (\langle x \trianglelefteq a \triangleright y \rangle \curvearrowright \alpha) =$ $(\ _{ctpb} (\langle x \rangle \curvearrowright \alpha) \trianglelefteq a \triangleright \ _{ctpb} (\langle y \rangle \curvearrowright \alpha)) \triangleleft ? a \triangleright (\mathbf{tau} \circ \ _{ctpb} (\alpha \curvearrowright \langle x \trianglelefteq a \triangleright y \rangle))$	Ctpb4
$\ _{ctpb} (\langle x \trianglelefteq \mathbf{swch} \triangleright y \rangle \curvearrowright \alpha) = \mathbf{tau} \circ (\ _{ctpb} (\alpha \curvearrowright \langle x \rangle) \triangleleft \alpha \neq \langle \rangle \triangleright \langle y \rangle)$	Ctpb5
$\ _{ctpb} (\langle x \trianglelefteq \mathbf{NT}(\beta) \triangleright y \rangle \curvearrowright \alpha) = \mathbf{tau} \circ \ _{ctpb} (\beta \curvearrowright \langle x \rangle \curvearrowright \alpha)$	Ctpb6
$\ _{ctpb} (\langle x \triangleleft ? a \triangleright y \rangle \curvearrowright \alpha) = \ _{ctpb} (\langle x \rangle \curvearrowright \alpha) \triangleleft ? a \triangleright \ _{ctpb} (\langle y \rangle \curvearrowright \alpha)$	Ctpb7

Table 5. Axioms for current thread persistence with blocking. Here $a \in \mathcal{BA}$.

are supposed to not to contain any guards. In our case, they threads can have the form $x \triangleleft ? a \triangleright y$ as seen in axiom Ctpb7.

The axioms in Table 5 are defined for finite threads only. For a sequence of arbitrary (finite or infinite) threads $\alpha = \langle p_1 \rangle \curvearrowright \dots \curvearrowright \langle p_m \rangle$, $\|_{ctpb} (\alpha)$ is determined by its projective sequence:

$$\pi_n(\|_{ctpb} (\alpha)) = \pi_n(\|_{ctpb} (\langle \pi_n(p_1) \rangle \curvearrowright \dots \curvearrowright \langle \pi_n(p_m) \rangle))$$

This can be obtained by applying the metric methodology of [3] as done for other strategic interleaving operators of thread algebra in [25, 26, 9].

We now denote \mathcal{T}_{svp} as the set of all closed terms over the signature of \mathbf{TA}_{svp} . The set \mathcal{B} of *basic terms* is inductively defined by the following rules:

- $S, D \in \mathcal{B}$;
- if $p \in \mathcal{B}$ then $\mathbf{tau} \circ p \in \mathcal{B}$;
- if $p, q \in \mathcal{B}$ and $a \in \mathcal{BA}$ then $p \trianglelefteq a \triangleright q \in \mathcal{B}$;
- if $p, q \in \mathcal{B}$ then $p \trianglelefteq \mathbf{swch} \triangleright q \in \mathcal{B}$;
- if $p, q, r_1, \dots, r_n \in \mathcal{B}$ then $p \trianglelefteq \mathbf{NT}(\langle r_1 \rangle \curvearrowright \dots \curvearrowright \langle r_n \rangle) \triangleright q \in \mathcal{B}$;
- if $p, q \in \mathcal{B}$ and $a \in \mathcal{BA}$ then $p \triangleleft \mathbf{y}_a \triangleright q \in \mathcal{B}$;
- if $p, q \in \mathcal{B}$ and $a \in \mathcal{BA}$ then $p \triangleleft ? a \triangleright q \in \mathcal{B}$;

Then \mathcal{B} is a subset of \mathcal{T}_{svp} . We write \mathcal{B}^0 for the set of all terms from \mathcal{B} in which no subterm of the form $p \trianglelefteq \mathbf{NT}(\alpha) \triangleright q$ occurs. We will show that each term from \mathcal{T}_{svp} can be reduced to a term from \mathcal{B} . We use the following lemma:

Lemma 1. *For all $p_1, \dots, p_n \in \mathcal{B}$, there is a term $q \in \mathcal{B}^0$ such that $\|_{ctpb} (\langle p_1 \rangle \curvearrowright \dots \curvearrowright \langle p_n \rangle) = q$ is derivable from the axioms of \mathbf{TA}_{svp} .*

Proof. This can be proven by induction on $\nu(p)$ where $\nu : \mathcal{T}_{svp} \rightarrow \mathbb{N}$ is recalled from [9] as follows:

$$\begin{aligned}
\nu(S) &= 1, \\
\nu(D) &= 1, \\
\nu(\mathbf{tau} \circ p) &= \nu(p) + 1, \\
\nu(p \trianglelefteq a \triangleright q) &= \nu(p) + \nu(q) + 1, \\
\nu(p \trianglelefteq \mathbf{NT}(\langle r_1 \rangle \curvearrowright \dots \curvearrowright \langle r_n \rangle) \triangleright q) &= \nu(p) + \nu(r_1) + \dots + \nu(r_n) + \nu(q) + 1, \\
\nu(p \triangleleft ? a \triangleright q) &= \nu(p) + \nu(q), \\
\nu(p \triangleleft \mathbf{y}_a \triangleright q) &= \nu(p) + \nu(q).
\end{aligned}$$

Theorem 1. (Elimination). *For all $p \in \mathcal{T}_{svp}$, there is a term $q \in \mathcal{B}$ such that $p = q$ is derivable from the axioms of TA_{svp} .*

Proof. We prove by induction on $\nu(p)$, where $\nu(p)$ is defined as in Lemma 1, and moreover:

$$\nu(\|_{ctpb} (\langle p_1 \rangle \curvearrowright \dots \curvearrowright \langle p_n \rangle)) = \nu(p_1) + \dots + \nu(p_n) + 1.$$

We consider the following cases:

- $p \in \{S, D\}$. Then $p \in \mathcal{B}$.
- $p = \mathbf{tau} \circ p'$. By the induction hypothesis, there is a basic term q' of \mathcal{B} such that $p' = q'$. Then $\mathbf{tau} \circ q' \in \mathcal{B}$. Hence $p = \mathbf{tau} \circ p' = \mathbf{tau} \circ q' \in \mathcal{B}$.
- $p = p' \triangleleft a \triangleright p''$. By the induction hypothesis, there are basic terms q' and q'' of \mathcal{B} such that $p' = q'$ and $p'' = q''$. Hence $p = p' \triangleleft a \triangleright p'' = q' \triangleleft a \triangleright q'' \in \mathcal{B}$.
- $p = p' \triangleleft \text{NT}(\langle r_1 \rangle \curvearrowright \dots \curvearrowright \langle r_n \rangle) \triangleright p''$. By the induction hypothesis, there are basic terms $q', r'_1, \dots, r'_n, q''$ of \mathcal{B} such that $p' = q', p'' = q''$ and $r_i = r'_i$ for $1 \leq i \leq n$. Hence $p = p' \triangleleft \text{NT}(\langle r_1 \rangle \curvearrowright \dots \curvearrowright \langle r_n \rangle) \triangleright p'' = q' \triangleleft \text{NT}(\langle r'_1 \rangle \curvearrowright \dots \curvearrowright \langle r'_n \rangle) \triangleright q'' \in \mathcal{B}$.
- $p = p' \triangleleft \mathbf{y}_a \triangleright p''$. Similar to the previous cases, $p \in \mathcal{B}$.
- $p = p' \triangleleft ?a \triangleright p''$. Similar to the previous cases, $p \in \mathcal{B}$.
- $p = \|_{ctpb} (\langle \rangle) = S \in \mathcal{B}$.
- $p = \|_{ctpb} (\langle p_1 \rangle \curvearrowright \dots \curvearrowright \langle p_n \rangle)$. By the induction hypothesis, there are basic terms q_1, \dots, q_n of \mathcal{B} such that $p_i = q_i$ for all $1 \leq i \leq n$. It follows from Lemma 1 that $p = \|_{ctpb} (\langle p_1 \rangle \curvearrowright \dots \curvearrowright \langle p_n \rangle) = \|_{ctpb} (\langle q_1 \rangle \curvearrowright \dots \curvearrowright \langle q_n \rangle) \in \mathcal{B}^0 \subseteq \mathcal{B}$.

We now write \mathcal{B}' for the set of all terms from \mathcal{B} in which no subterm of the form $p \triangleleft c \triangleright q$ occurs where c is already a guard of $p \triangleleft c \triangleright q$. Furthermore, if $p \triangleleft c \triangleright q \in \mathcal{B}'$ then $p = q$ cannot be derivable from the axioms TA_{svp} .

Lemma 2. *For all $p \in \mathcal{B}$, there is a term $q \in \mathcal{B}'$ such that $p = q$ is derivable from the axiom of TA_{svp} .*

Proof. This can be proven by induction on $l(p) : \mathcal{B} \rightarrow \mathbb{N}$ using the axioms in Table 1 where $l(p)$ is defined inductively as follows:

$$\begin{aligned} l(S) &= 1, \\ l(D) &= 1, \\ l(p \triangleleft a \triangleright q) &= \max\{l(p), l(q)\} + 1, \\ l(p \triangleleft c \triangleright q) &= \max\{l(r) \mid r \in R_{p \triangleleft c \triangleright q}\} + |G_{p \triangleleft c \triangleright q}| \end{aligned}$$

where $R_{p \triangleleft c \triangleright q}$ and $G_{p \triangleleft c \triangleright q}$ are the sets of real residuals and guards of $p \triangleleft c \triangleright q$.

Corollary 1. *For all $p \in \mathcal{T}_{svp}$, there is a term $q \in \mathcal{B}'$ such that $p = q$ is derivable from the axioms of TA_{svp} .*

$\xi \xi'$	$= \xi' \xi$
$(\xi \xi') \xi''$	$= \xi (\xi' \xi'')$
$\mathbf{tau}\xi$	$= \xi$
$?(\xi \xi')$	$= ?\xi \wedge ?\xi'$
$x \triangleleft \mathbf{y}_{\xi \xi'} \triangleright y = x$	

Table 6. Conditions on the synchronization function.

4 Synchronous cooperation of threads in SVP

In this section, we extend TA_{svp} with a form of synchronous cooperation of threads in SVP. The synchronous cooperation of threads in TA has been considered before in [9]. In this strategy, the current actions from all concurrent threads are simultaneously performed. If a thread is blocked then the whole system will be blocked. In SVP, we intend to perform simultaneously the maximum number of independent actions from the concurrent threads. This will avoid the communication-deadlock in the case that one of the threads is blocked, and might speed up processors [24]. We call this strategy *synchronous cooperation with blocking*.

4.1 Atomic actions and concurrent actions

Like [9], we assume a fixed but arbitrary set \mathcal{AA} of *atomic actions*, a fixed but arbitrary set $\mathcal{CA} \supseteq \mathcal{AA}$ of *concurrent actions*, and a fixed but arbitrary synchronization function $| : \mathcal{CA} \times \mathcal{CA} \rightarrow \mathcal{CA}$ satisfying that:

- $\mathbf{tau} \in \mathcal{AA}$;
- for an action $\xi \in \mathcal{CA}$ if and only if $\xi \in \mathcal{AA}$ or there exist ξ', ξ'' such that $\xi = \xi'|\xi''$;
- for an action $\xi \in \mathcal{CA}$ there is a boolean value $?\xi$ stating that ξ is independent or blocked;
- the equations in Table 6 are also satisfied with $\xi, \xi', \xi'' \in \mathcal{CA}$.

Hence, each concurrent action can be reduced to one of the following form:

- a with $a \in \mathcal{AA}$;
- $a_1|\dots|a_n$ with $a_1, \dots, a_n \in \mathcal{AA}$ for $n > 1$;

The set \mathcal{BA} of basic actions is extended with this set \mathcal{CA} of concurrent actions. We note that the last axiom of Table 6 on reply conditions states that the execution of the act of simultaneously performing produces a positive reply.

We write $|_{i \in I} \xi_i$ to denote the simultaneous action of the concurrent actions ξ_i with $i \in I$. Furthermore, $\wedge_{i \in \{i_1, \dots, i_k\}} ?\xi_i = ?\xi_{i_1} \wedge ?\xi_{i_2} \wedge \dots \wedge ?\xi_{i_k}$.

$\ _{scb} (\langle \rangle) = S$	Scb1
$\ _{scb} (\alpha \curvearrowright \langle S \rangle \curvearrowright \beta) = \ _{scb} (\alpha \curvearrowright \beta)$	Scb2
$\ _{scb} (\alpha \curvearrowright \langle D \rangle \curvearrowright \beta) = D$	Scb3
$\ _{scb} (\langle x_1 \trianglelefteq \xi_1 \triangleright y_1 \rangle \curvearrowright \dots \curvearrowright \langle x_n \trianglelefteq \xi_n \triangleright y_n \rangle) =$ $\quad _{i \in I} \xi_i \circ \ _{scb} (\langle \chi_i^I(x_1 \trianglelefteq \xi_1 \triangleright y_1) \rangle \curvearrowright \dots \curvearrowright \langle \chi_n^I(x_n \trianglelefteq \xi_n \triangleright y_n) \rangle)$ $\quad \text{where } I \subseteq [1, n]: \wedge_{i \in I} ?\xi_i \wedge \nexists j \notin I : ?\xi_j$	Scb4
$\ _{scb} (\alpha \curvearrowright \langle x \trianglelefteq \mathbf{NT}(\langle z_1 \rangle \curvearrowright \dots \curvearrowright \langle z_n \rangle) \triangleright y \rangle \curvearrowright \beta) =$ $\quad \ _{scb} (\alpha \curvearrowright \langle \mathbf{tau} \circ z_1 \rangle \curvearrowright \dots \curvearrowright \langle \mathbf{tau} \circ z_n \rangle \curvearrowright \langle \mathbf{tau} \circ x \rangle \curvearrowright \beta)$	Scb5
$\ _{scb} (\alpha \curvearrowright \langle x \triangleleft c \triangleright y \rangle \curvearrowright \beta) = \ _{scb} (\alpha \curvearrowright \langle x \rangle \curvearrowright \beta) \triangleleft c \triangleright \ _{scb} (\alpha \curvearrowright \langle y \rangle \curvearrowright \beta)$	Scb6

Table 7. Axioms for synchronous cooperation with blocking.

4.2 The synchronous cooperation with blocking strategy

The axioms for synchronous cooperation with blocking are given in Table 7. In this strategy, the composition of an empty sequence of threads is a termination. If a thread is in deadlock then the whole system is also in deadlock. If a thread is terminated then this thread is simply removed from the sequence. If all threads are deadlock free, the synchronous cooperation strategy will execute simultaneously all and only independent threads. The indexes of these threads are contained in I . We note that the \mathbf{tau} actions will arise when a family of thread is created in order to keep pace with other threads in the sequence. Furthermore, the threads are supposed initially not to contain any guards. The auxiliary function χ_i^I is defined by:

$$\chi_i^I(x \trianglelefteq \xi \triangleright y) = \begin{cases} x \triangleleft y_\xi \triangleright y & \text{if } i \in I, \\ x \trianglelefteq \xi \triangleright y & \text{otherwise.} \end{cases}$$

The axioms in Table 7 are defined for finite threads only. Like the $\|_{ctpb}$ operator, for a sequence of arbitrary (finite or infinite) threads $\alpha = \langle p_1 \rangle \curvearrowright \dots \curvearrowright \langle p_m \rangle$, $\|_{scb}(\alpha)$ is determined by its projective sequence:

$$\pi_n(\|_{scb}(\alpha)) = \pi_n(\|_{scb}(\langle \pi_n(p_1) \rangle \curvearrowright \dots \curvearrowright \langle \pi_n(p_m) \rangle))$$

To show that a term from \mathcal{T}_{svp} can still be reduced to a term from \mathcal{B} , we provide the following result:

Lemma 3. *For all $p_1, \dots, p_n \in \mathcal{B}$, there is a term $q \in \mathcal{B}^0$ such that $\|_{scb}(\langle p_1 \rangle \curvearrowright \dots \curvearrowright \langle p_n \rangle) = q$ is derivable from the axioms of \mathbf{TA}_{svp} .*

Proof. This can be proven by induction on $\nu(p)$ where $\nu(p)$ is defined as in Lemma 1 and Theorem 1 except that

$$\nu(p \trianglelefteq \mathbf{NT}(\langle r_1 \rangle \curvearrowright \dots \curvearrowright \langle r_n \rangle) \triangleright q) = \nu(p) + \nu(r_1) + \dots + \nu(r_n) + \nu(q) + n + 2.$$

Hence, one can see that Theorem 1 still holds for \mathbf{TA}_{svp} extended with the $\|_{scb}$ operator.

$\langle x_i E \rangle = t_i(\langle x_1 E \rangle, \dots, \langle x_n E \rangle) \ (i \in [1, n])$	RDP
If $y_i = t_i(y_1, \dots, y_n)$ for $i \in [1, n]$ then $y_i = \langle x_i E \rangle \ (i \in [1, n])$	RSP

Table 8. Axioms for the constants $\langle X | E \rangle$.

4.3 Guarded recursive specifications in TA_{svp}

In this section, we adapt the definition of guarded recursion in [9] to define guarded recursive specifications for TA_{svp} .

We assume the existence of a fixed but arbitrary set of variables \mathcal{X} . Let $X \subseteq \mathcal{X}$. We write \mathcal{T}_{svp}^X for the set of all terms from \mathcal{T}_{svp} in which no other variables than the ones in X have free occurrences.

The set \mathcal{G} of *guarded* terms is defined inductively as follows:

- $S, D \in \mathcal{G}$;
- if $\xi \in \mathcal{BA}$ and $t_1, t_2 \in \mathcal{T}_{svp}$ then $t_1 \triangleleft \xi \triangleright t_2 \in \mathcal{G}$;
- if $t_1, t_2, t_3 \in \mathcal{T}_{svp}$ then $t_1 \triangleleft \text{NT}(t_3) \triangleright t_2 \in \mathcal{G}$;
- if $\xi \in \mathcal{BA}$ and $t_1, t_2 \in \mathcal{G}$ then $t_1 \triangleleft y_\xi \triangleright t_2 \in \mathcal{G}$;
- if $\xi \in \mathcal{BA}$ and $t_1, t_2 \in \mathcal{G}$ then $t_1 \triangleleft ?\xi \triangleright t_2 \in \mathcal{G}$;
- if $t_1, \dots, t_n \in \mathcal{G}$ then $\|_{ctpb} (\langle t_1 \rangle \curvearrowright \dots \curvearrowright \langle t_n \rangle) \in \mathcal{G}$.
- if $t_1, \dots, t_n \in \mathcal{G}$ then $\|_{scb} (\langle t_1 \rangle \curvearrowright \dots \curvearrowright \langle t_n \rangle) \in \mathcal{G}$.

A *finite recursive specification* E is a finite set $\{x_i = t_i \mid i \in [1, n]\}$ of recursive equations where t_i , for all $1 \leq i \leq n$, are terms in $\mathcal{T}_{svp}^{\{x_1, \dots, x_n\}}$. The finite recursive specification E is *guarded* if for all $1 \leq i \leq n$, t_i are guarded.

Theorem 2. *A guarded recursive specification determines a unique solution in A^∞ .*

Proof. The proof of this theorem can be obtained in the same line as the proof of Theorem 5 in [9] in the terminology of metric topology.

If E is a guarded recursive specification and x a recursive variable in E , then $\langle x | E \rangle$ denotes the thread that has to be substituted for x in the solution for E . This thread is called *regular*. The axioms for guarded recursive specifications are given in Table 8, where RDP and RSP refer to *Recursive Definition Principle* and *Recursive Specification Principle* as in other process algebras (see e.g. [14]).

5 Structural operational semantics for TA_{svp}

Structural operational semantics (SOS) [23, 1] is a formal semantics of programming languages. This section presents a SOS for TA_{svp} . We will show that the multithreads obtained via $\|_{ctpb}$ and $\|_{scb}$ operators are communication-deadlock free. Furthermore, the axioms of TA_{svp} are sound and complete with respect to the bisimulation equivalence induced by this SOS.

$\langle S, \rho \rangle \downarrow$	$\langle D, \rho \rangle \uparrow$	$\langle x \triangleleft \mathbf{tau} \triangleright y, \rho \rangle \xrightarrow{\mathbf{tau}} \langle x, \rho \rangle$	$\langle x \triangleleft \mathbf{NT}(\beta) \triangleright y, \rho \rangle \xrightarrow{\mathbf{NT}(\beta)} \langle x, \rho \rangle$
$\frac{\xi \in \rho(\langle \rangle), \rho(\xi)(\xi) = T}{\langle x \triangleleft \xi \triangleright y, \rho \rangle \xrightarrow{\xi} \langle x, \frac{\partial}{\partial \xi} \rho \rangle}$	$\frac{\xi \in \rho(\langle \rangle), \rho(\xi)(\xi) = F}{\langle x \triangleleft \xi \triangleright y, \rho \rangle \xrightarrow{\xi} \langle y, \frac{\partial}{\partial \xi} \rho \rangle}$	$\frac{\xi \notin \rho(\langle \rangle)}{\langle x \triangleleft \xi \triangleright y, \rho \rangle \downarrow}$	
$\frac{\langle x, \rho \rangle \downarrow, \xi \in \rho(\langle \rangle)}{\langle x \triangleleft ? \xi \triangleright y, \rho \rangle \downarrow}$	$\frac{\langle x, \rho \rangle \uparrow, \xi \in \rho(\langle \rangle)}{\langle x \triangleleft ? \xi \triangleright y, \rho \rangle \uparrow}$	$\frac{\langle x, \rho \rangle \downarrow, \xi \in \rho(\langle \rangle)}{\langle x \triangleleft ? \xi \triangleright y, \rho \rangle \downarrow}$	$\frac{\langle x, \rho \rangle \xrightarrow{\xi'} \langle x', \rho' \rangle, \xi \in \rho(\langle \rangle)}{\langle x \triangleleft ? \xi \triangleright y, \rho \rangle \xrightarrow{\xi'} \langle x', \rho' \rangle}$
$\frac{\langle x, \rho \rangle \downarrow, \xi \notin \rho(\langle \rangle)}{\langle x \triangleleft ? \xi \triangleright y, \rho \rangle \downarrow}$	$\frac{\langle x, \rho \rangle \uparrow, \xi \notin \rho(\langle \rangle)}{\langle x \triangleleft ? \xi \triangleright y, \rho \rangle \uparrow}$	$\frac{\langle y, \rho \rangle \downarrow, \xi \notin \rho(\langle \rangle)}{\langle x \triangleleft ? \xi \triangleright y, \rho \rangle \downarrow}$	$\frac{\langle x, \rho \rangle \xrightarrow{\xi'} \langle x', \rho' \rangle, \xi \notin \rho(\langle \rangle)}{\langle x \triangleleft ? \xi \triangleright y, \rho \rangle \xrightarrow{\xi'} \langle x', \rho' \rangle}$
$\frac{\langle x, \rho \rangle \downarrow, \rho(\langle \rangle)(\xi) = T}{\langle x \triangleleft \mathbf{y}_\xi \triangleright y, \rho \rangle \downarrow}$	$\frac{\langle x, \rho \rangle \uparrow, \rho(\langle \rangle)(\xi) = T}{\langle x \triangleleft \mathbf{y}_\xi \triangleright y, \rho \rangle \uparrow}$	$\frac{\langle x, \rho \rangle \downarrow, \rho(\langle \rangle)(\xi) = T}{\langle x \triangleleft \mathbf{y}_\xi \triangleright y, \rho \rangle \downarrow}$	
$\frac{\langle x, \rho \rangle \xrightarrow{\xi'} \langle x', \rho' \rangle, \rho(\langle \rangle)(\xi) = T}{\langle x \triangleleft \mathbf{y}_\xi \triangleright y, \rho \rangle \xrightarrow{\xi'} \langle x', \rho' \rangle}$			
$\frac{\langle y, \rho \rangle \downarrow, \rho(\langle \rangle)(\xi) = F}{\langle x \triangleleft \mathbf{y}_\xi \triangleright y, \rho \rangle \downarrow}$	$\frac{\langle y, \rho \rangle \uparrow, \rho(\langle \rangle)(\xi) = F}{\langle x \triangleleft \mathbf{y}_\xi \triangleright y, \rho \rangle \uparrow}$	$\frac{\langle y, \rho \rangle \downarrow, \rho(\langle \rangle)(\xi) = F}{\langle x \triangleleft \mathbf{y}_\xi \triangleright y, \rho \rangle \downarrow}$	
$\frac{\langle y, \rho \rangle \xrightarrow{\xi'} \langle y', \rho' \rangle, \rho(\langle \rangle)(\xi) = F}{\langle x \triangleleft \mathbf{y}_\xi \triangleright y, \rho \rangle \xrightarrow{\xi'} \langle y', \rho' \rangle}$			

Table 9. Transition rules for BTA_{svp} . Here $\xi \in \mathcal{BA} \setminus \{\mathbf{tau}, \mathbf{NT}(\beta)\}$.

5.1 Transition rules for BTA_{svp}

We adapt the SOS given in [9] that deals with blocking for TA_{svp} . In this SOS, each action can be accepted or rejected by the execution environment, depending on the execution history of the thread and external conditions. Let $\rho : \mathcal{BA}^* \rightarrow (\mathcal{BA} \rightarrow \{T, F\})$ be a function representing an execution environment. Given an execution environment ρ and an action $\xi \in \mathcal{BA}$, ξ is accepted by ρ if $\xi \in \rho(\langle \rangle)$. Note that in TA_{svp} , ξ is always accepted by ρ , i.e. $\xi \in \rho(\langle \rangle)$, if all predecessors of the thread containing ξ are terminated. If ξ is accepted, the derived execution environment of ρ after processing ξ is defined by $\frac{\partial}{\partial \xi} \rho(\alpha) = \rho(\langle \xi \rangle \curvearrowright \alpha)$.

The following transition relations on threads are used in the SOS of TA_{svp} :

- the *action step* $\langle p, \rho \rangle \xrightarrow{\xi} \langle p', \rho' \rangle$ for each $\xi \in \mathcal{BA}$;
- the *action step* $\langle p, \rho \rangle \xrightarrow{\mathbf{NT}(\alpha)} \langle p', \rho' \rangle$: in execution environment ρ , thread p can fork off a family of threads α and after that proceeds as thread p' in execution environment ρ' ;
- the *termination* $\langle p, \rho \rangle \downarrow$;
- the *deadlock* $\langle p, \rho \rangle \uparrow$; and
- the *blocking* $\langle p, \rho \rangle \downarrow$;

Transition rules for BTA_{svp} are described in Table 9. Transition rules for approximation operators and guarded recursion are given in Table 10.

$\frac{\langle x, \rho \rangle \downarrow}{\langle \pi_{n+1}(x), \rho \rangle \downarrow}$	$\frac{\langle x, \rho \rangle \uparrow}{\langle \pi_0(x), \rho \rangle \uparrow}$	$\frac{\langle x, \rho \rangle \uparrow}{\langle \pi_{n+1}(x), \rho \rangle \uparrow}$	$\frac{\langle x, \rho \rangle \downarrow}{\langle \pi_{n+1}(x), \rho \rangle \downarrow}$	$\frac{\langle x, \rho \rangle \xrightarrow{\xi} \langle x', \rho' \rangle}{\langle \pi_{n+1}(x), \rho \rangle \xrightarrow{\xi} \langle \pi_n(x'), \rho' \rangle}$
$\frac{\langle \langle t E \rangle, \rho \rangle \downarrow}{\langle \langle x E \rangle, \rho \rangle \downarrow} x = t \in E$	$\frac{\langle \langle t E \rangle, \rho \rangle \uparrow}{\langle \langle x E \rangle, \rho \rangle \uparrow} x = t \in E$	$\frac{\langle \langle t E \rangle, \rho \rangle \downarrow}{\langle \langle x E \rangle, \rho \rangle \downarrow} x = t \in E$		
$\frac{\langle \langle t E \rangle, \rho \rangle \xrightarrow{\xi} \langle x', \rho' \rangle}{\langle \langle x E \rangle, \rho \rangle \xrightarrow{\xi} \langle x', \rho' \rangle} x = t \in E$				

Table 10. Transition rules for approximation operators and guarded recursion.

$\frac{\langle x_1, \rho \rangle \downarrow, \dots, \langle x_n, \rho \rangle \downarrow}{\langle \parallel_{ctpb} (\langle x_1 \rangle \curvearrowright \dots \curvearrowright \langle x_n \rangle), \rho \rangle \downarrow}$	$\frac{\langle x_1, \rho \rangle \uparrow}{\langle \parallel_{ctpb} (\langle x_1 \rangle \curvearrowright \dots \curvearrowright \langle x_n \rangle), \rho \rangle \uparrow}$
$\frac{\langle x_1, \rho \rangle \downarrow, \dots, \langle x_n, \rho \rangle \downarrow, \langle x_{n+1}, \rho \rangle \xrightarrow{a} \langle x'_{n+1}, \rho' \rangle}{\langle \parallel_{ctpb} (\langle x_1 \rangle \curvearrowright \dots \curvearrowright \langle x_n \rangle \curvearrowright \langle x_{n+1} \rangle \curvearrowright \alpha), \rho \rangle \xrightarrow{a} \langle \parallel_{ctpb} (\langle x'_{n+1} \rangle \curvearrowright \alpha), \rho' \rangle}$	
$\frac{\langle x_1, \rho \rangle \downarrow, \dots, \langle x_n, \rho \rangle \downarrow, \langle x_{n+1}, \rho \rangle \downarrow}{\langle \parallel_{ctpb} (\langle x_1 \rangle \curvearrowright \dots \curvearrowright \langle x_n \rangle \curvearrowright \langle x_{n+1} \rangle \curvearrowright \alpha), \rho \rangle \xrightarrow{\mathbf{tau}} \langle \parallel_{ctpb} (\alpha \curvearrowright \langle x_{n+1} \rangle), \rho \rangle}$	
$\frac{\langle x_1, \rho \rangle \downarrow, \dots, \langle x_n, \rho \rangle \downarrow, \alpha \neq \langle \rangle}{\langle \parallel_{ctpb} (\langle x_1 \rangle \curvearrowright \dots \curvearrowright \langle x_n \rangle \curvearrowright \langle x \leq \mathbf{swch} \geq y \rangle \curvearrowright \alpha), \rho \rangle \xrightarrow{\mathbf{tau}} \langle \parallel_{ctpb} (\alpha \curvearrowright \langle x \rangle), \rho \rangle}$	
$\frac{\langle x_1, \rho \rangle \downarrow, \dots, \langle x_n, \rho \rangle \downarrow}{\langle \parallel_{ctpb} (\langle x_1 \rangle \curvearrowright \dots \curvearrowright \langle x_n \rangle \curvearrowright \langle x \leq \mathbf{swch} \geq y \rangle), \rho \rangle \xrightarrow{\mathbf{tau}} \langle \parallel_{ctpb} \langle y \rangle, \rho \rangle}$	
$\frac{\langle x_1, \rho \rangle \downarrow, \dots, \langle x_n, \rho \rangle \downarrow, \langle x_{n+1}, \rho \rangle \xrightarrow{\mathbf{NT}(\beta)} \langle x'_{n+1}, \rho \rangle}{\langle \parallel_{ctpb} (\langle x_1 \rangle \curvearrowright \dots \curvearrowright \langle x_n \rangle \curvearrowright \langle x_{n+1} \rangle \curvearrowright \alpha), \rho \rangle \xrightarrow{\mathbf{tau}} \langle \parallel_{ctpb} (\beta \curvearrowright \langle x'_{n+1} \rangle \curvearrowright \alpha), \rho \rangle}$	

Table 11. Transition rules for current thread persistence with blocking. Here $a \in \mathcal{BA} \setminus \{\mathbf{swch}, \mathbf{NT}(\beta)\}$.

5.2 Transition rules for current thread persistence with blocking.

Transition rules for the \parallel_{ctpb} operator are given in Table 11.

Theorem 3. (Communication-deadlock free for \parallel_{ctpb}). *A multithread obtained via the current thread persistence with blocking operator is communication-deadlock free, i.e. given an execution environment, the multithread either terminates or is in deadlock or there is a thread in the sequence that can execute its current action after some (zero or more) \mathbf{tau} steps.*

Proof. Let $p = \parallel_{ctpb} (\langle p_1 \rangle \curvearrowright \dots \curvearrowright \langle p_n \rangle)$. If $p = S$ or $p = D$ then we are done. In the remaining case, there is a thread $p_i \neq S$ such that it has no predecessors or its predecessors are terminated. Then either $\langle p_i, \rho \rangle \uparrow$ or $\langle p_i, \rho \rangle \xrightarrow{a_i} \langle p'_i, \rho' \rangle$ for every execution environment ρ since the blocking of a thread depends only on its predecessors. Hence, according to the transition rules in Table 11, the

$\frac{\langle x_1, \rho \rangle \downarrow, \dots, \langle x_n, \rho \rangle \downarrow}{\langle \parallel_{scb} (\langle x_1 \rangle \curvearrowright \dots \curvearrowright \langle x_n \rangle), \rho \rangle \downarrow} \quad \frac{\langle x_i, \rho \rangle \uparrow}{\langle \parallel_{scb} (\langle x_1 \rangle \curvearrowright \dots \curvearrowright \langle x_n \rangle), \rho \rangle \uparrow} \quad 1 \leq i \leq n$
$\frac{\{\langle x_j, \rho \rangle \downarrow \mid j \in J\}, \{\langle x_k, \rho \rangle \downarrow \mid k \in K\}, J \cup K = [1, n]}{\langle \parallel_{scb} (\langle x_1 \rangle \curvearrowright \dots \curvearrowright \langle x_n \rangle), \rho \rangle \downarrow}$
$\frac{\{ \langle x_i, \rho \rangle \xrightarrow{\xi_i} \langle x'_i, \rho'_i \rangle \mid i \in I \}, \{\langle x_j, \rho \rangle \downarrow \mid j \in J\}, \{\langle x_k, \rho \rangle \downarrow \mid k \in K\},}{I \neq \emptyset, I \cup J \cup K = [1, n]}$
$\langle \parallel_{scb} (\langle x_1 \rangle \curvearrowright \dots \curvearrowright \langle x_n \rangle), \rho \rangle \xrightarrow{ i \in I \xi'_i} \langle \parallel_{scb} (\alpha_{b(1)} \curvearrowright \dots \curvearrowright \alpha_{b(I \cup K)}), \frac{\partial}{\partial i \in I \xi'_i} \rho \rangle$
<p>where $\alpha_m = \begin{cases} \langle x_m \rangle & \text{if } m \in K, \\ \chi_m^I(x_m) & \text{and } \xi'_m = \xi_m \text{ if } \xi_m \neq \mathbf{NT}(\beta), \\ \beta \curvearrowright \langle x'_m \rangle & \text{and } \xi'_m = \mathbf{tau} \text{ if } \xi_m = \mathbf{NT}(\beta) \end{cases}$</p>

Table 12. Transition rules for synchronous cooperation of threads.

multithread will be in deadlock or one of threads can execute its current action after a number ($\leq i$) **tau**-steps.

5.3 Transition rules for the synchronous cooperation with blocking

Transition rules for the \parallel_{scb} operator are given in Table 12. In this table, we use an auxiliary function b which stands for an arbitrary bijective function from $[1, |I \cup K|]$ to $I \cup K$ such that for all $m \in [1, |I \cup K|]$, $b(m) \leq b(|I \cup K|)$.

Theorem 4. (Communication-deadlock free for \parallel_{scb}). *A multithread obtained via the synchronous cooperation with blocking strategy is communication-deadlock free, i.e. it is not blocked.*

Proof. If this thread is a termination or a deadlock then we are done. In the remaining case, the proof follows from the fact that there always exists the set $I \neq \emptyset$ of all independent actions. It is because there always exists an active thread (that is not a termination neither a deadlock) whose current action is not blocked (see the proof of Theorem 3).

5.4 Bisimulation

A *bisimulation* is a symmetric binary relation B on closed terms such that for all closed terms p and q :

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

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

The above bisimulation equivalence is given in [9], and is the *stateless* bisimilarity according to [22]. It is a congruence with respect to the transition rules given in Table 9, Table 11 and Table 12 since these transition rules are in the relaxed panth format [19]. The transition labels containing terms, as stated in [9], do not complicate matters because there are no volatile involved [21].

Theorem 5. (Soundness). *The axioms given in Table 1, Table 2, Table 3, Table 4, Table 5, Table 6, Table 7, and Table 8 are sound with respect to bisimulation equivalence.*

Proof. Since bisimulation is both an equivalence and congruence with respect to TA_{svp} , we only need to check that if $s = t$ is an axiom in TA_{svp} and σ is a closed substitution then $\sigma(s) \Leftrightarrow \sigma(t)$. We consider the following axioms:

- Ctpb2: $\langle \sigma(\|_{\text{ctpb}} (\langle S \rangle \curvearrowright \alpha)), \rho \rangle$ and $\langle \sigma(\|_{\text{ctpb}} (\alpha)), \rho \rangle$ will perform the same initial transitions to the same states as $\langle \|_{\text{ctpb}} (\langle p \rangle \curvearrowright \alpha'), \rho \rangle$, where $p \neq S$ and $\sigma(\alpha) = \underbrace{\langle S \rangle \curvearrowright \dots \curvearrowright \langle S \rangle}_{\geq 0} \curvearrowright \langle p \rangle \curvearrowright \alpha'$. Thus,

$$\sigma(\|_{\text{ctpb}} (\langle S \rangle \curvearrowright \alpha)) \stackrel{\geq 0}{\Leftrightarrow} \sigma(\|_{\text{ctpb}} (\alpha)).$$
- Ctpb3: $\langle \sigma(\|_{\text{ctpb}} (\langle D \rangle \curvearrowright \alpha)), \rho \rangle \uparrow$ and $\langle D, \rho \rangle \uparrow$. Thus, $\sigma(\|_{\text{ctpb}} (\langle D \rangle \curvearrowright \alpha)) \Leftrightarrow D$.
- Ctpb4: Let s and t be the left- and right-hand sides of this axiom. Then

$$\langle \sigma(s), \rho \rangle \xrightarrow{\text{tau}} \langle \sigma(\|_{\text{ctpb}} (\alpha \curvearrowright \langle x \leq a \triangleright y \rangle)), \rho \rangle$$
 and

$$\langle \sigma(t), \rho \rangle \xrightarrow{\text{tau}} \langle \sigma(\|_{\text{ctpb}} (\alpha \curvearrowright \langle x \leq a \triangleright y \rangle)), \rho \rangle$$
 if $a \notin \rho(\langle \rangle)$. Otherwise,

$$\langle \sigma(s), \rho \rangle \xrightarrow{a} \langle \sigma(\|_{\text{ctpb}} (\langle x \rangle \curvearrowright \alpha)), \frac{\partial}{\partial a} \rho \rangle$$
 and

$$\langle \sigma(t), \rho \rangle \xrightarrow{a} \langle \sigma(\|_{\text{ctpb}} (\langle x \rangle \curvearrowright \alpha)), \frac{\partial}{\partial a} \rho \rangle$$
 if $\rho(a)(a) = T$, and

$$\langle \sigma(s), \rho \rangle \xrightarrow{a} \langle \sigma(\|_{\text{ctpb}} (\langle y \rangle \curvearrowright \alpha)), \frac{\partial}{\partial a} \rho \rangle$$
 and

$$\langle \sigma(t), \rho \rangle \xrightarrow{a} \langle \sigma(\|_{\text{ctpb}} (\langle y \rangle \curvearrowright \alpha)), \frac{\partial}{\partial a} \rho \rangle$$
 if $\rho(a)(a) = F$. Hence, $\sigma(s) \Leftrightarrow \sigma(t)$.
- Ctpb5-7 are checked similarly as the previous axiom.
- Scb2-6 are checked similarly as the axioms Ctpb2-6.
- RDP, RSP and AIP are checked as in other process algebras (see e.g. [14]).
- Other axioms are trivial.

Theorem 6. (Ground-completeness). *The axioms given in Table 1, Table 2, Table 3, Table 4, Table 5, Table 6, Table 7, and Table 8 are complete with respect to bisimulation equivalence.*

Proof. Let p and q be two regular threads. We show that if $p \Leftrightarrow q$ then $p = q$ can be derived from the axioms of TA_{svp} .

- First, we show that if p and q are closed terms (or finite) then $p = q$ is derivable from the axioms of TA_{svp} . It follows from Corollary 1 that p and q can be reduced to basic terms p' and q' in \mathcal{B}' . Hence, $p = p'$ and $q = q'$. Since $p \Leftrightarrow q$, it follows from Theorem 5 that $p' \Leftrightarrow q'$. One can prove by induction on $\max\{l(p'), l(q')\}$ that $p' = q'$ is derivable from the axioms of TA_{svp} . Therefore $p = q$ is derivable from the axioms of TA_{svp} if p and q are finite.

- We now assume that p and q are solutions of two guarded recursive specifications. Since bisimulation is a congruence, $\pi_n(p) \Leftrightarrow \pi_n(q)$ for all $n \in \mathbb{N}$. It follows from the previous case that $\pi_n(p) = \pi_n(q)$ for all $n \in \mathbb{N}$. By AIP, $p = q$.

6 Conclusion

In this paper, we have taken thread algebra as a theoretical framework to the study of the SANE Virtual Processor (SVP), an architectural model for effectively programming distributed multiprocessor systems.

We have formalized SVP program behaviors and two interleaving strategies that are used in SVP in the setting of TA_{svp} (thread algebra for SVP). We have presented a projective limit model as a denotational semantics, and a structural operational semantics (SOS) for TA_{svp} . We have shown that SVP programs has a desired property, i.e. they are communication-deadlock free. Furthermore, the axioms of TA_{svp} are sound and complete with respect to bisimulation equivalence induced by this SOS. In the future, we will concern with determinism of SVP programs, i.e., the program should always give the same result, a key property of the sequential paradigm.

References

1. 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 197–222. Elsevier, 2001.
2. J.C.M. Baeten and J.A. Bergstra. Process algebra with signals and conditions. In M. Broy, editor, *Programming and Mathematical Methods*, volume F88 of *NATO ASI Series*, pages 1–21, 1992.
3. J.W. Bakker and J.I. Zucker. Processes and the denotational semantics of concurrency. *Information and Control*, 54(1/2):70–120, 1982.
4. J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Inform. and Control*, 60 (1-3):109–137, 1984.
5. J.A. Bergstra and M.E. Loots. Program algebra for sequential code. *J. of Logic and Algebraic Programming*, 51:125–156, 2002.
6. J.A. Bergstra and C.A. Middelburg. Thread algebra for strategic interleaving. Computing Science Report PRG0404, Programming Research Group, University of Amsterdam, 2004.
7. J.A. Bergstra and C.A. Middelburg. Simulating turing machines on maurer machines. CS-Report 05-28, Department of mathematics and computer science, Technische Universiteit Eindhoven, 2005.
8. J.A. Bergstra and C.A. Middelburg. Maurer computers for pipelined instruction processing. CS-Report 06-12, Department of mathematics and computer science, Technische Universiteit Eindhoven, 2006.
9. J.A. Bergstra and C.A. Middelburg. Synchronous cooperation for explicit multi-threading. CS-Report 06-29, Department of mathematics and computer science, Technische Universiteit Eindhoven, 2006.

10. J.A. Bergstra and C.A. Middelburg. Maurer computers with single-thread control. *Fundamenta Informaticae*, 2007. To appear.
11. J.A. Bergstra and C.A. Middelburg. Thread algebra for strategic interleaving. *Formal Aspects of Computing*, 2007. To appear. Preliminary version: Computer Science Report PRG0404: Sectie Software Engineering, University of Amsterdam.
12. A. Bolychevsky, C.R. Jesshope, and V. Muchnick. Dynamic scheduling in risc architectures. In *IEE Proceedings Computers and Digital Techniques*, volume 143(5), pages 309–317, 1996.
13. K. Bousias, N.M. Hasasneh, and C.R. Jesshope. Instruction-level parallelism through Microthreading—a scalable Approach to chip multiprocessors. *The Computer Journal*, 49 (2):211–233, 2006.
14. W.J. Fokink. *Introduction to Process Algebra*. Texts in Theoretical Computer Science, An EATCS Series. Springer, 2000.
15. C.R. Jesshope. SVP and μ TC-A dynamic model of concurrency and its implementation as a compiler target. <http://staff.science.uva.nl/jesshope/Papers/uTCPaper.pdf>.
16. C.R. Jesshope. Multithreaded microprocessors evolution or revolution. In Sedukhin Omondo, editor, *ACSAC 2003: Advances in Computer Systems Architecture*, pages 21–45, 2003.
17. C.R. Jesshope. Microthreading a model for distributed instruction-level concurrency. *Parallel Processing Letters*, 16 (2):209–228, 2006.
18. C.R. Jesshope and B. Luo. Micro-threading: A new approach to future risc. In *ACAC 2000*, pages 31–41. IEEE Computer Society Press, 2000.
19. C.A. Middelburg. An alternative formulation of operational conservativity with binding terms. *Journal of Logic and Algebraic Programming*, 55(1/2):1–19, 2003.
20. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
21. M.R. Mousavi, M.J. Gabbay, and M.A. Reniers. Sos for higher order processes. In M. Abadi and L. de Alfaro, editors, *CONCUR 2005*, pages 308–322, 2005.
22. M.R. Mousavi, M.A. Reniers, and J.F. Groote. Congruence for sos with data. In *LICS'04*, pages 303–312, 2004.
23. G. Plotkin. A structural approach to operational semantics. Aarhus DAIMI FN-19, Computing Science Department, 1981.
24. T. Ungerer, B. Robič, and J. Šilc. A survey of processors with explicit multithreading. *ACM Computing Surveys*, 35 (1):29–63, 2003.
25. T.D. Vu. Denotational semantics for thread algebra. *Journal of Logic and Algebraic Programming*. To appear.
26. T.D. Vu. *Semantics and applications of process and program algebra*. PhD thesis, University of Amsterdam, 2007.