Acta Informatica manuscript No.
(will be inserted by the editor)

J.A. Bergstra · C.A. Middelburg

Synchronous Cooperation for Explicit Multi-Threading

the date of receipt and acceptance should be inserted later

Abstract We develop an algebraic theory of threads, synchronous cooperation of threads and interaction of threads with Maurer machines, and investigate program parallelization using the resulting theory. Program parallelization underlies techniques for speeding up instruction processing on a computer that make use of the abilities of the computer to process instructions simultaneously in cases where the state changes involved do no influence each other. One of our findings is that a strong induction principle is needed when proving theorems about sufficient conditions for the correctness of program parallelizations. The induction principle introduced has brought us to construct a projective limit model for the theory developed.

Keywords thread algebra – synchronous cooperation – program algebra – program parallelization – projective limit model

1 Introduction

Thread algebra originates from the form of process algebra introduced in [6] under the name basic polarized process algebra. A thread is the behaviour of a deterministic sequential program under execution. In earlier work, see e.g. [7,14,13], we

The work presented in this paper has been partly carried out while the second author was also at Eindhoven University of Technology, Department of Mathematics and Computer Science.

The work presented in this paper has been carried out as part of the GLANCE-project MICROGRIDS, which is funded by the Netherlands Organisation for Scientific Research (NWO).

J.A. Bergstra · C.A. Middelburg
Programming Research Group, University of Amsterdam, P.O. Box 41882, 1009 DB Amsterdam, the Netherlands
E-mail: J.A.Bergstra@uva.nl, C.A.Middelburg@uva.nl

J.A. Bergstra
Department of Philosophy, Utrecht University, P.O. Box 80126, 3508 TC Utrecht, the Netherlands

have elaborated forms of concurrency where the actions to be performed by the different threads involved are interleaved according to some deterministic interleaving strategy. Synchronous cooperation is the form of concurrency where at each stage the actions to be performed by the different threads involved are all performed simultaneously. In the current paper, we develop an algebraic theory of threads, synchronous cooperation of threads and interaction of threads with Maurer machines. We call the resulting theory a thread algebra for synchronous cooperation.

Threads can be used to direct a Maurer machine in performing operations on its state. Maurer machines are based on a model for computers proposed by Maurer in [23]. Maurer's model for computers is quite different from the well-known models for computers in theoretical computer science such as register machines, multi-stack machines and Turing machines (see e.g. [20]). The strength of Maurer's model is that it is close to real computers. Maurer's model is based on the view that a computer has a memory, the contents of all memory elements make up the state of the computer, the computer processes instructions, and the processing of an instruction amounts to performing an operation on the state of the computer which results in changes of the contents of certain memory elements.

Explicit multi-threading is a basic technique to speed up instruction processing by a machine (see e.g. [29]). Explicit multi-threading techniques require that programs are parallelized by judicious use of forking. In this paper, we investigate program parallelization for simple programs without test and jump instructions using the thread algebra for synchronous cooperation developed and program algebra.

Program algebra is introduced in [5, 6]. In program algebra, not the behaviour of deterministic sequential programs under execution is considered, but the programs themselves. A program is viewed as an instruction sequence. The behaviour of a program is taken for a thread of the kind considered in thread algebra. Program algebra provides a program notation which is close to existing assembly languages.

By employing the thread algebra for synchronous cooperation developed to investigate program parallelization, we demonstrate that this thread algebra has at least one interesting application. On the other hand, setting up a framework in which program parallelization can be investigated, is one of the objectives with which we have developed a thread algebra for synchronous cooperation. For that very reason, we have chosen to use Maurer's model for computers. Unlike this relatively unknown model, the well-known models for computers in theoretical computer science have little in common with real computers. They abstract from many aspects of real computers which must be taken into account when investigating program parallelization.

In earlier work on thread algebra, synchronous cooperation was not considered. To deal with synchronous cooperation in thread algebra, we introduce in the thread algebra for synchronous cooperation a special action (δ) which blocks threads. This feature was not present in earlier work on thread algebra. We also introduce another feature that was not present in earlier work on thread algebra, namely conditional action repetition. In modelling instruction processing, this feature is convenient to deal with instructions of which the processing on a computer takes more than one step. Typical examples of such instructions are load instruc-

tions, which may even take many steps in case of cache misses. Moreover, we introduce the notions of state transformer equivalence and computation. Both notions are relevant to program parallelization: if two threads are state transformer equivalent, then the computations directed by those threads beginning in the same initial state terminate in the same final state, but they may have different lengths.

One of the findings of our investigation of program parallelization is that a strong induction principle is needed when proving theorems about sufficient conditions for the correctness of program parallelizations. Therefore, we introduce an induction principle to establish state transformer equivalence of infinite threads. This induction principle is based on the view that any infinite thread is fully characterized by the infinite sequence of all its finite approximations. The model that we construct for the thread algebra for synchronous cooperation, including the above-mentioned induction principle, is a projective limit model (see e.g. [4, 22]) because such a model fits in very well with this view.

In addition to the thread algebra for synchronous cooperation, we use a simple variant of the program algebra from [6] to investigate program parallelization. This simple variant offers a convenient notation for studying program parallelization: the programs concerned permit a direct analysis of semantic issues involved. It covers only simple programs without test and jump instructions. This is a drastic simplification. Because of the complexity of program parallelization, we consider a simplification like this one desirable to start with.

We regard the work presented in this paper, like the preceding work presented in [8–10], as a preparatory step in developing, as part of a project investigating micro-threading [16, 21], a formal approach to design new micro-architectures. That approach should allow for the correctness of new micro-architectures and their anticipated speed-up results to be verified.

The structure of this paper is as follows. First, we develop most of the thread algebra for synchronous cooperation (Section 2). Next, we present a projective limit model for the thread algebra developed so far (Section 3). Then, we complete the thread algebra developed so far with an operator for applying a thread to a Maurer machine from one of its states and introduce the notion of computation in the resulting setting (Section 4). Following this, we introduce the notion of state transformer equivalence of threads and give some state transformer properties of threads (Section 5). After that, we present the simple variant of program algebra and introduce classes of program relevant to the investigation of program parallelization (Section 6). Next, we investigate program parallelization, focused on finding sufficient conditions for the correctness of program parallelizations (Section 7). Finally, we make some concluding remarks (Section 8). Appendix B contains a glossary of symbols used in this paper.

In Section 3, some familiarity with metric spaces is assumed. The definitions of all notions concerning metric spaces that are assumed known in those sections can be found in most introductory textbooks on topology. We mention [17] as an example of an introductory textbook in which those notions are introduced in an intuitively appealing way.

2 Thread Algebra for Synchronous Cooperation

In this section, we develop most of the thread algebra for synchronous cooperation used in the investigation of program parallelization later on. First, we treat the kernel of the thread algebra in question. Next, we add step by step several features, including synchronous cooperation and conditional action repetition, to the kernel. Finally, we present a structural operational semantics for the thread algebra developed in this section.

2.1 Basic Thread Algebra with Blocking

BTA_δ (Basic Thread Algebra with Blocking) is a form of process algebra which is tailored to the description of the behaviour of deterministic sequential programs under execution. The behaviours concerned are called *threads*.

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

- the *deadlock* constant D;
- the *termination* constant S;
- for each $a \in \mathscr{A}_\delta$, 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 over the signature of BTA_δ, abbreviates $p \trianglelefteq a \trianglerighteq p$.

The intuition is that each basic action performed by a thread is taken as a command to be processed by the execution environment of the thread. 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 thread concerned. Let p and q be closed terms over the signature of BTA_δ and $a \in \mathscr{A}$. Then $p \trianglelefteq a \trianglerighteq q$ will perform action a, and after that proceed as p if the processing of a leads to the reply T (called a positive reply) and proceed as q if the processing of a leads to the reply F (called a negative reply). The action tau plays a special role: its processing will never change any state and always lead to a positive reply. The action δ blocks a thread: the execution environment cannot process it and consequently a reply value is never returned. Hence, $p \trianglelefteq \delta \trianglerighteq q$ cannot but become inactive, just as D.

Example 1 Consider the term $\mathit{inc} \circ (\mathsf{S} \trianglelefteq \mathit{dec} \trianglerighteq \mathsf{D})$ and an execution environment in which processing of basic actions *inc* and *dec* amounts to incrementing and decrementing a counter by one. Suppose that the counter concerned can take only non-negative values. Furthermore, suppose that the processing of *inc* leads always to a positive reply and the processing of *dec* leads to a positive reply if the value of the counter is not zero and to a negative reply otherwise. In this execution environment, $\mathit{inc} \circ (\mathsf{S} \trianglelefteq \mathit{dec} \trianglerighteq \mathsf{D})$ will first perform *inc*, next perform *dec*, and then terminate. It will not deadlock instead of terminate because the value of the counter will be greater than zero when *dec* is performed.

The axioms of BTA_δ are given in Table 1. Using the abbreviation introduced above, axiom T1 can be written as follows: $x \trianglelefteq \mathsf{tau} \trianglerighteq y = \mathsf{tau} \circ x$.

Table 1 Axioms of BTA_δ

$x \trianglelefteq \mathsf{tau} \trianglerighteq y = x \trianglelefteq \mathsf{tau} \trianglerighteq x$	T1
$x \trianglelefteq \delta \trianglerighteq y = \mathsf{D}$	T2

Table 2 Conditions on the synchronization function

$(\xi \,\&\, \xi') \,\&\, \xi'' = \xi \,\&\, (\xi' \,\&\, \xi'')$
$(\xi \,\&\, \xi') \,\&\, \xi'' = (\xi' \,\&\, \xi) \,\&\, \xi''$
$\mathsf{tau} \,\&\, \xi = \xi$
$\delta \,\&\, \xi = \delta$
$\xi \,\&\, \delta = \delta$

2.2 Synchronous Cooperation of Threads

We extend BTA_δ with a form of synchronous cooperation that supports thread forking. The result is called $\mathrm{TA}_{\mathrm{sc}}$. Synchronous cooperation requires the introduction of atomic actions and concurrent actions.

In $\mathrm{TA}_{\mathrm{sc}}$, it is assumed that there are a fixed but arbitrary set $\mathscr{CA}_\delta$ of *concurrent actions*, a fixed but arbitrary finite set $\mathscr{AA} \subseteq \mathscr{CA}_\delta$ of *atomic actions* and a fixed but arbitrary *synchronization* function $\& : \mathscr{CA}_\delta \times \mathscr{CA}_\delta \to \mathscr{CA}_\delta$ such that:

- $\mathsf{tau} \in \mathscr{AA}$ and $\delta \notin \mathscr{AA}$;
- $\xi \in \mathscr{CA}_\delta$ iff $\xi = \delta$ or $\xi \in \mathscr{AA}$ or there exist $\xi', \xi'' \in \mathscr{CA}_\delta$ such that $\xi = \xi' \,\&\, \xi''$;
- for all $\xi, \xi', \xi'' \in \mathscr{CA}_\delta$, the equations given in Table 2 are satisfied.

It is further assumed that $\mathscr{A}_\delta = \mathscr{CA}_\delta$. We write $\mathscr{CA}$ for $\mathscr{CA}_\delta \setminus \{\delta\}$.

A concurrent action $\xi \,\&\, \xi'$, where $\xi, \xi' \in \mathscr{CA}$, represents the act of simultaneously performing ξ and ξ' unless $\xi \,\&\, \xi' = \delta$. Concurrent actions ξ and ξ' for which $\xi \,\&\, \xi' = \delta$ are regarded to be actions for which the act of simultaneously performing them is impossible.

It is not assumed that & satisfies $\xi \,\&\, \xi' = \xi' \,\&\, \xi$, for all $\xi, \xi' \in \mathscr{CA}_\delta$, because one of the axioms of $\mathrm{TA}_{\mathrm{sc}}$ introduced below (axiom RC2) entails that $\xi \,\&\, \xi'$ and $\xi' \,\&\, \xi$ can lead to different replies. The assumption that $\mathscr{AA}$ is finite has a technical background. Only the results presented in Appendix A depend on it.

Using the equations of Table 2, each concurrent action can be reduced to one of the following three forms:

- δ;
- a with $a \in \mathscr{AA}$;
- $a_1 \,\&\, \ldots \,\&\, a_n$ with $a_1, \ldots, a_n \in \mathscr{AA}$ $(n > 1)$.

The concurrent action $a_1 \,\&\, \ldots \,\&\, a_n$, where $a_1, \ldots, a_n \in \mathscr{AA}$, represents the act of simultaneously performing the atomic actions $a_1, \ldots, a_n$.

A collection of threads that proceed concurrently is assumed to take the form of a sequence, called a thread vector. Synchronous cooperation is the form of concurrency where at each stage the actions to be performed by the different threads in the thread vector are all performed simultaneously. In earlier work, see e.g. [7, 14,

13], we have elaborated forms of concurrency where the actions to be performed by the different threads involved are interleaved according to some deterministic interleaving strategy. In that work, we have also elaborated several interleaving strategies that support thread forking. All of them deal with imperfect forking, i.e. forking off a thread may be blocked and/or may fail. In this paper, we cover only perfect forking. We believe that perfect forking is a suitable abstraction when studying program parallelization. Unless capacity problems arise with regard to forking, it needs not block or fail. We believe that software tools responsible for program parallelization should see to it that such capacity problems will never arise.

$\mathrm{TA}_{\mathrm{sc}}$ has the constants and operators of BTA_δ and in addition the following operators:

- the unary *synchronous cooperation* operator $\|^{\mathsf{s}}$;
- the ternary *forking postconditional composition* operator $\_ \trianglelefteq \mathsf{nt}(\_) \trianglerighteq \_$;
- for each $\xi \in \mathscr{CA}_\delta$, a binary *reply conditional* operator $\_ \lhd \mathsf{y}_\xi \rhd \_$.

The synchronous cooperation operator is a unary operator of which the operand denotes a sequence of threads. Like action prefixing, we introduce *forking prefixing* as an abbreviation: $\mathsf{nt}(p) \circ q$, where p and q are terms over the signature of $\mathrm{TA}_{\mathrm{sc}}$, abbreviates $q \trianglelefteq \mathsf{nt}(p) \trianglerighteq q$. Henceforth, the postconditional composition operators introduced in Section 2.1 will be called non-forking postconditional composition operators.

The forking postconditional composition operator has the same shape as non-forking postconditional composition operators. Formally, no action is involved in forking postconditional composition. However, for an operational intuition, in $p \trianglelefteq \mathsf{nt}(r) \trianglerighteq q$, $\mathsf{nt}(r)$ can be considered a thread forking action. It represents the act of forking off thread r. Like with real actions, a reply is produced. We consider the case where forking off a thread will never be blocked or fail. In that case, it always produces a positive reply. The action tau arises as a residue in both the thread forking off a thread and the thread being forked off. In that way, those threads keep pace with the other threads that proceed concurrently. In [7], $\mathsf{nt}(r)$ was formally considered a thread forking action. We experienced afterwards that this leads to unnecessary complications in expressing definitions and results concerning the projective limit model for the thread algebra developed in this paper (see Section 3).

The reply conditional operators $\_ \lhd \mathsf{y}_\xi \rhd \_$ are auxiliary operators needed to deal properly with the replies produced for actions that are performed simultaneously on account of synchronous cooperation of threads. Suppose that $\xi_1 \& \ldots \& \xi_n$ is the last action performed. Let p and q be closed terms over the signature of $\mathrm{TA}_{\mathrm{sc}}$, and let $\xi \in \{\xi_1, \ldots, \xi_n\}$. Then $p \lhd \mathsf{y}_\xi \rhd q$ behaves as p if processing of ξ alone would have led to the reply T and it behaves as q if processing of ξ alone would have led to the reply F. The case where $\xi \notin \{\xi_1, \ldots, \xi_n\}$ is irrelevant to synchronous cooperation. Nothing is stipulated about the behaviour of $p \lhd \mathsf{y}_\xi \rhd q$ in this case. In fact, it may differ from one execution environment to another.

The axioms for synchronous cooperation with perfect forking are given in Table 3.[1] In this table, $\xi_1, \ldots, \xi_n$ and ξ stand for arbitrary members of $\mathscr{CA}_\delta$. The

[1] 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 finite sequences α and β. We assume the usual laws for concatenation of finite sequences.

Table 3 Axioms for synchronous cooperation with perfect forking

$\|^{\mathsf{s}}(\langle\rangle) = \mathsf{S}$	SCf1
$\|^{\mathsf{s}}(\alpha \frown \langle \mathsf{S} \rangle \frown \beta) = \|^{\mathsf{s}}(\alpha \frown \beta)$	SCf2
$\|^{\mathsf{s}}(\alpha \frown \langle \mathsf{D} \rangle \frown \beta) = \mathsf{D}$	SCf3
$\|^{\mathsf{s}}(\langle x_1 \trianglelefteq \xi_1 \trianglerighteq y_1 \rangle \frown \ldots \frown \langle x_n \trianglelefteq \xi_n \trianglerighteq y_n \rangle) =$ $\xi_1 \,\&\, \ldots \,\&\, \xi_n \circ \|^{\mathsf{s}}(\langle x_1 \lhd \mathsf{y}_{\xi_1} \rhd y_1 \rangle \frown \ldots \frown \langle x_n \lhd \mathsf{y}_{\xi_n} \rhd y_n \rangle)$	SCf4
$\|^{\mathsf{s}}(\alpha \frown \langle x \trianglelefteq \mathsf{nt}(z) \trianglerighteq y \rangle \frown \beta) = \|^{\mathsf{s}}(\alpha \frown \langle \mathsf{tau} \circ x \rangle \frown \langle \mathsf{tau} \circ z \rangle \frown \beta)$	SCf5
$\|^{\mathsf{s}}(\alpha \frown \langle x \lhd \mathsf{y}_\xi \rhd y \rangle \frown \beta) = \|^{\mathsf{s}}(\alpha \frown \langle x \rangle \frown \beta) \lhd \mathsf{y}_\xi \rhd \|^{\mathsf{s}}(\alpha \frown \langle y \rangle \frown \beta)$	SCf6

Table 4 Axioms for reply conditionals

$x \trianglelefteq \xi \trianglerighteq y = \xi \circ (x \lhd \mathsf{y}_\xi \rhd y)$	RC1
$\xi \,\&\, \xi' \neq \delta \Rightarrow x \lhd \mathsf{y}_{\xi \& \xi'} \rhd y = x \lhd \mathsf{y}_{\xi'} \rhd y$	RC2
$x \lhd \mathsf{y}_{\mathsf{tau}} \rhd y = x$	RC3
$x \lhd \mathsf{y}_\delta \rhd y = x$	RC4
$x \lhd \mathsf{y}_a \rhd x = x$	RC5
$(x \lhd \mathsf{y}_a \rhd y) \lhd \mathsf{y}_a \rhd z = x \lhd \mathsf{y}_a \rhd z$	RC6
$x \lhd \mathsf{y}_a \rhd (y \lhd \mathsf{y}_a \rhd z) = x \lhd \mathsf{y}_a \rhd z$	RC7
$(x \lhd \mathsf{y}_a \rhd y) \lhd \mathsf{y}_b \rhd z = (x \lhd \mathsf{y}_b \rhd z) \lhd \mathsf{y}_a \rhd (y \lhd \mathsf{y}_b \rhd z)$	RC8
$x \lhd \mathsf{y}_a \rhd (y \lhd \mathsf{y}_b \rhd z) = (x \lhd \mathsf{y}_a \rhd y) \lhd \mathsf{y}_b \rhd (x \lhd \mathsf{y}_a \rhd z)$	RC9

axioms for reply conditionals are given in Table 4. In this table, ξ and ξ' stand for arbitrary members of $\mathscr{CA}_\delta$ and a and b stand for arbitrary members of $\mathscr{AA}$.

The crucial axioms for synchronous cooperation with perfect forking are axioms SCf4 and SCf5. Axiom SCf4 expresses that, in the case where each thread in the thread vector can perform an action, first the actions to be performed by the different threads are all performed simultaneously and after that the synchronous cooperation proceeds as if the actions performed by the different threads were performed alone. Axiom SCf5 expresses that, in the case where some threads in the thread vector can fork off a thread, forking off threads takes place such that the threads forking off a thread and the threads being forked off keep pace with the other threads in the thread vector. The crucial axiom for reply conditionals is axiom RC1. This axiom expresses that the behaviour of a reply conditional for the last action performed is determined by the reply to which the processing of that action has led.

Axiom RC2 reflects that, for ξ and ξ' such that $\xi \,\&\, \xi' \neq \delta$, the reply to which the processing of $\xi \,\&\, \xi'$ leads is the reply to which the processing of ξ' leads. An alternative to axiom RC2 is

$$\xi \,\&\, \xi' \neq \delta \Rightarrow x \lhd \mathsf{y}_{\xi \& \xi'} \rhd y = (x \lhd \mathsf{y}_\xi \rhd y) \lhd \mathsf{y}_{\xi'} \rhd y \,,$$

which reflects that, for ξ and ξ' such that $\xi \,\&\, \xi' \neq \delta$, the reply to which the processing of $\xi \,\&\, \xi'$ leads is the conjunction of the reply to which the processing of ξ leads and the reply to which the processing of ξ' leads. This alternative would result in a slightly different theory. Both axiom RC2 and the alternative are plau-

sible, but we believe that the alternative would complicate the investigation of program parallelization slightly.

Axiom RC4 looks odd: δ blocks a thread because it does not lead to any reply. Axiom RC4 stipulates that a reply conditional for δ behaves as if blocking of a thread leads to a positive reply. An alternative to axiom RC4 is

$$x \trianglelefteq \mathsf{y}_\delta \trianglerighteq y = y \,,$$

which stipulates that a reply conditional for δ behaves as if blocking of a thread leads to a negative reply. The choice between axiom RC4 and this alternative makes little difference: each occurrence of a reply conditional for δ introduced by applying axioms of $\mathrm{TA}_{\mathrm{sc}}$ is always a subterm of a term that is derivably equal to D.

Example 2 Consider the term $\|^{\mathsf{s}}(\langle \mathit{inc}_1 \circ \mathsf{S}\rangle \frown \langle \mathit{inc}_2 \circ \mathsf{S}\rangle)$, which according to the axioms of $\mathrm{TA}_{\mathrm{sc}}$ equals $\mathit{inc}_1 \,\&\, \mathit{inc}_2 \circ \mathsf{S}$. Take the synchronization function $\&$ such that $\mathit{inc}_1 \,\&\, \mathit{inc}_2 \neq \delta$, which amounts to assuming that each execution environment can process inc_1 and inc_2 at the same time. Then, in any execution environment, $\|^{\mathsf{s}}(\langle \mathit{inc}_1 \circ \mathsf{S}\rangle \frown \langle \mathit{inc}_2 \circ \mathsf{S}\rangle)$ will first perform inc_1 and inc_2 simultaneously and then terminate. In an execution environment as described in Example 1, but now with two counters, simultaneously performing inc_1 and inc_2 results in incrementing two counters at once. Notice that the term $\|^{\mathsf{s}}(\langle \mathsf{nt}(\mathit{inc}_2 \circ \mathsf{S}) \circ (\mathit{inc}_1 \circ \mathsf{S})\rangle)$, which involves thread forking, equals $\mathsf{tau} \circ \|^{\mathsf{s}}(\langle \mathit{inc}_1 \circ \mathsf{S}\rangle \frown \langle \mathit{inc}_2 \circ \mathsf{S}\rangle)$.

Henceforth, we write $\mathscr{T}_{\mathrm{TA}_{\mathrm{sc}}}$ for the set of all closed terms over the signature of $\mathrm{TA}_{\mathrm{sc}}$.

The set $\mathscr{B}$ of *basic terms* is inductively defined by the following rules:

- $\mathsf{S}, \mathsf{D} \in \mathscr{B}$;
- if $p \in \mathscr{B}$, then $\mathsf{tau} \circ p \in \mathscr{B}$;
- if $\xi \in \mathscr{B}\!\mathscr{A}$ and $p, q \in \mathscr{B}$, then $p \trianglelefteq \xi \trianglerighteq q \in \mathscr{B}$;
- if $p, q, r \in \mathscr{B}$, then $p \trianglelefteq \mathsf{nt}(r) \trianglerighteq q \in \mathscr{B}$;
- if $\xi \in \mathscr{B}\!\mathscr{A}$ and $p, q \in \mathscr{B}$, then $p \trianglelefteq \mathsf{y}_\xi \trianglerighteq q \in \mathscr{B}$.

We write $\mathscr{B}^0$ for the set of all terms from $\mathscr{B}$ in which no subterm of the form $p \trianglelefteq \mathsf{nt}(r) \trianglerighteq q$ occurs. Clearly, $\mathscr{B}$ is a subset of $\mathscr{T}_{\mathrm{TA}_{\mathrm{sc}}}$. Each term from $\mathscr{T}_{\mathrm{TA}_{\mathrm{sc}}}$ can be reduced to a term from $\mathscr{B}$.

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

Proof The proof follows a similar line as the proof of Theorem 2 from [14]. This means that it is a proof by induction on the structure of p in which some cases boil down to proving a lemma by some form of induction or another, mostly again structural induction. Here, we have to consider the additional case $p \equiv p' \trianglelefteq \mathsf{y}_\xi \trianglerighteq p''$, where we can restrict ourselves to basic terms p' and p''. This case is easily proved using axioms RC3 and RC4. Moreover, the case $p \equiv \|^{\mathsf{s}}(\langle p'_1\rangle \frown \ldots \frown \langle p'_n\rangle)$, where we can restrict ourselves to basic terms $p'_1, \ldots, p'_n$, cannot be proved by induction on the sum of the depths plus one of $p'_1, \ldots, p'_n$ and case distinction on

the structure of p'_1. Instead, it is proved by induction on $\nu(p)$, where $\nu : \mathscr{T}_{\mathrm{TA}_{\mathrm{sc}}} \to \mathbb{N}$ is defined by

$$\begin{aligned}
&\nu(\mathsf{S}) = 1\,,\\
&\nu(\mathsf{D}) = 1\,,\\
&\nu(\mathsf{tau} \circ p) = \nu(p) + 1\,,\\
&\nu(p \trianglelefteq \xi \trianglerighteq q) = \nu(p) + \nu(q) + 1 \qquad \text{if } \xi \neq \mathsf{tau}\,,\\
&\nu(p \trianglelefteq \mathsf{nt}(r) \trianglerighteq q) = \nu(p) + \nu(r) + 3\,,\\
&\nu(p \triangleleft \mathsf{y}_\xi \triangleright q) = \nu(p) + \nu(q)\,,\\
&\nu(\|^{\mathsf{s}}(\langle p_1\rangle \frown \ldots \frown \langle p_n\rangle)) = \nu(p_1) + \ldots + \nu(p_n) + 1\,,
\end{aligned}$$

and case distinction according to the left-hand sides of the axioms for synchronous cooperation, which yields an exhaustive case distinction. The proofs for the different cases go similar. We sketch here the proof for the case corresponding to the left-hand side of axiom SCf5. It is the case where $p'_i \equiv p'' \trianglelefteq \mathsf{nt}(r'') \trianglerighteq q''$ for some $i \in [1,n]$. In this case, if follows from axiom SCf5 and the definition of ν that there exists a term p' such that $p = p'$ is derivable from the axioms of $\mathrm{TA}_{\mathrm{sc}}$ and $\nu(p) = \nu(p') + 1$. Because $p = p'$ and $\nu(p) > \nu(p')$, it follows immediately from the induction hypothesis that there exists a term $q \in \mathscr{B}$ such that $p = q$ is derivable from the axioms of $\mathrm{TA}_{\mathrm{sc}}$. □

The function ν defined in the proof of Theorem 1 is used in coming proofs as well. The following is a useful corollary from the proof of Theorem 1.

Corollary 1 *For all $p_1, \ldots, p_n \in \mathscr{B}$, there exists a term $q \in \mathscr{B}^0$ such that $\|^{\mathsf{s}}(\langle p_1\rangle \frown \ldots \frown \langle p_n\rangle) = q$ is derivable from the axioms of* $\mathrm{TA}_{\mathrm{sc}}$.

This corollary implies that each closed term from $\mathscr{T}_{\mathrm{TA}_{\mathrm{sc}}}$ in which all subterms of the form $p \trianglelefteq \mathsf{nt}(r) \trianglerighteq q$ occur in a subterm of the form $\|^{\mathsf{s}}(\langle p_1\rangle \frown \ldots \frown \langle p_n\rangle)$, can be reduced to a term from $\mathscr{B}$ in which no subterm of the form $p \trianglelefteq \mathsf{nt}(r) \trianglerighteq q$ occurs.

The following lemma will be used in the proof of Proposition 13.

Lemma 1 *Let $p_0 \in \mathscr{B}^0$, and let $p_1, \ldots, p_n \in \mathscr{B}$. Then $\|^{\mathsf{s}}(\langle p_0\rangle \frown \ldots \frown \langle p_n\rangle) = \|^{\mathsf{s}}(\langle p_0\rangle \frown \langle \|^{\mathsf{s}}(\langle p_1\rangle \frown \ldots \frown \langle p_n\rangle)\rangle)$.*

Proof This is straightforwardly proved by induction on the structure of p_0, and in the case $p_0 \equiv p' \trianglelefteq \xi \trianglerighteq p''$ by induction on $\nu(p_1) + \ldots + \nu(p_n)$ and case distinction according to the left-hand side of the axioms for synchronous cooperation. Moreover, in the case $p_0 \equiv \mathsf{S}$, it has to be proved that $\|^{\mathsf{s}}(\langle p_1\rangle \frown \ldots \frown \langle p_n\rangle) = \|^{\mathsf{s}}(\langle \|^{\mathsf{s}}(\langle p_1\rangle \frown \ldots \frown \langle p_n\rangle)\rangle)$. This is proved similarly. □

We have taken the operator $\|^{\mathsf{s}}$ for a unary operator of which the operand denotes a sequence of threads. This matches well with the intuition that synchronous cooperation operates on a thread vector. We can look upon the operator $\|^{\mathsf{s}}$ as if there is actually an n-ary operator, of which the operands denote threads, for every $n \in \mathbb{N}$. In Section 3, we will look upon the operator $\|^{\mathsf{s}}$ in this way for the purpose of more concise expression of definitions and results concerning the projective limit model for the thread algebra developed in this paper.

Table 5 Axioms for conditional action repetition

$\xi\,{}^{*\mathsf{T}}\,x = x \trianglelefteq \xi \trianglerighteq (\xi\,{}^{*\mathsf{T}}\,x)$	CAR1
$\xi\,{}^{*\mathsf{F}}\,x = (\xi\,{}^{*\mathsf{F}}\,x) \trianglelefteq \xi \trianglerighteq x$	CAR2

2.3 Conditional Action Repetition

We extend $\mathrm{TA}_{\mathrm{sc}}$ with conditional action repetition. The result is called $\mathrm{TA}_{\mathrm{sc}}^*$.

We add, for each $\xi \in \mathscr{A}_\delta$ and $b \in \{\mathsf{T},\mathsf{F}\}$, a unary *conditional action repetition* operator $\xi\,{}^{*b}\,\_$ to $\mathrm{TA}_{\mathrm{sc}}$. Let p be a closed term over the signature of $\mathrm{TA}_{\mathrm{sc}}^*$. Then $\xi\,{}^{*\mathsf{T}}\,p$ performs ξ as many times as needed for a positive reply, and then proceeds as p. In the case of $\xi\,{}^{*\mathsf{F}}\,p$, the role of the reply is reversed. The axioms for conditional action repetition are given in Table 5. In this table, ξ stands for an arbitrary member of $\mathscr{A}_\delta$.

Example 3 Consider the term $dec\,{}^{*\mathsf{F}}\,(inc \circ \mathsf{S})$ and an execution environment as described in Example 1. In this execution environment, $dec\,{}^{*\mathsf{F}}\,(inc \circ \mathsf{S})$ will first perform *dec* as many times as needed for a negative reply, next perform *inc*, and then terminate. At the moment of termination, the value of the counter will be one because the processing of *dec* will lead to a negative reply only when the counter is zero.

We introduce *split-action prefixing* as an abbreviation: $\xi/\xi' \circ p$, where p is a term over the signature of $\mathrm{TA}_{\mathrm{sc}}^*$ and $\xi,\xi' \in \mathscr{A}_\delta$, abbreviates $p \trianglelefteq \xi \trianglerighteq (\xi'\,{}^{*\mathsf{T}}\,p)$. This means that $\xi/\xi' \circ p$ performs ξ once and next ξ' as many times as needed for a positive reply, and then proceeds as p. If the processing of ξ produces a positive reply, then ξ' is not at all performed.

Henceforth, we write $\mathscr{T}_{\mathrm{TA}_{\mathrm{sc}}^*}$ for the set of all closed terms over the signature of $\mathrm{TA}_{\mathrm{sc}}^*$.

Below, we introduce a subset $\mathscr{C}$ of $\mathscr{T}_{\mathrm{TA}_{\mathrm{sc}}^*}$ which is reminiscent of $\mathscr{B}$. The significance of $\mathscr{C}$ is that several properties that need to be proved for all terms from some subset of $\mathscr{C}$ can be proved for all terms from $\mathscr{C}$ by structural induction in a straightforward manner.

The set $\mathscr{C}$ of *semi-basic terms* is inductively defined by the following rules:

- $\mathsf{S},\mathsf{D} \in \mathscr{C}$;
- if $p \in \mathscr{C}$, then $\mathsf{tau} \circ p \in \mathscr{C}$;
- if $\xi \in \mathscr{B}\mathscr{A}$ and $p,q \in \mathscr{C}$, then $p \trianglelefteq \xi \trianglerighteq q \in \mathscr{C}$;
- if $p,q,r \in \mathscr{C}$, then $p \trianglelefteq \mathsf{nt}(r) \trianglerighteq q \in \mathscr{C}$;
- if $\xi \in \mathscr{B}\mathscr{A}$ and $p,q \in \mathscr{C}$, then $p \triangleleft \mathsf{y}_\xi \triangleright q \in \mathscr{C}$;
- if $\xi \in \mathscr{B}\mathscr{A}$ and $p \in \mathscr{C}$, then $\xi\,{}^{*\mathsf{T}}\,p \in \mathscr{C}$ and $\xi\,{}^{*\mathsf{F}}\,p \in \mathscr{C}$.

We write $\mathscr{C}^0$ for the set of all terms from $\mathscr{C}$ in which no subterm of the form $p \trianglelefteq \mathsf{nt}(r) \trianglerighteq q$ occurs. Clearly, $\mathscr{B}$ is a subset of $\mathscr{C}$ and $\mathscr{C}$ is a subset of $\mathscr{T}_{\mathrm{TA}_{\mathrm{sc}}^*}$. Terms from $\mathscr{C}$ with a subterm of the form $\xi\,{}^{*\mathsf{T}}\,p$ or the form $\xi\,{}^{*\mathsf{F}}\,p$ cannot be reduced to terms from $\mathscr{B}$. The projection operators introduced in Section 2.4 enable a kind of approximate reduction for terms from $\mathscr{C}$.

We write $p \cdot q$, where $p \in \mathscr{C}^0$ and $q \in \mathscr{T}_{\mathrm{TA}_{\mathrm{sc}}^*}$, for p with each occurrence of S replaced by q. On purpose, this notation is suggestive of sequential composition.

Table 6 Approximation induction principle

$\bigwedge_{n\geq 0}\pi_n(x)=\pi_n(y)\Rightarrow x=y$ AIP

Table 7 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
$\pi_{n+1}(x\trianglelefteq\xi\trianglerighteq y)=\pi_n(x)\trianglelefteq\xi\trianglerighteq\pi_n(y)$	P3
$\pi_{n+1}(x\trianglelefteq\mathsf{nt}(z)\trianglerighteq y)=\pi_n(x)\trianglelefteq\mathsf{nt}(\pi_n(z))\trianglerighteq\pi_n(y)$	P4
$\pi_{n+1}(x\triangleleft\mathsf{y}_\xi\triangleright y)=\pi_{n+1}(x)\triangleleft\mathsf{y}_\xi\triangleright\pi_{n+1}(y)$	P5

However, we use $\cdot$ to denote a syntactic operation, i.e. an operation on terms. This notation will turn out to be convenient when formulating properties relevant to program parallelization.

2.4 Approximation Induction Principle

Each closed term over the signature of $\mathrm{TA}_{\mathrm{sc}}$ denotes a finite thread, i.e. a thread of which the length of the sequences of actions that it can perform is bounded. However, not each closed term over the signature of $\mathrm{TA}^*_{\mathrm{sc}}$ denotes a finite thread: conditional action repetition gives rise to infinite threads. Closed terms over the signature of $\mathrm{TA}^*_{\mathrm{sc}}$ that denote the same infinite thread cannot always be proved equal by means of the axioms of $\mathrm{TA}^*_{\mathrm{sc}}$. We introduce the approximation induction principle to reason about infinite threads.

The approximation induction principle, AIP in short, is based on the view that two threads are identical if their approximations up to any finite depth are identical. The approximation up to depth n of a thread is obtained by cutting it off after performing a sequence of actions of length n.

AIP is the infinitary conditional equation given in Table 6. Here, following [6], approximation of depth n is phrased in terms of a unary *projection* operator π_n. The projection operators are defined inductively by means of the axioms given in Table 7. In this table, ξ stands for an arbitrary member of $\mathscr{A}$.

Let $p\in\mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$. Then it follows from AIP that:

$$x=p\trianglelefteq\xi\trianglerighteq x\Rightarrow x=\xi\,{}^{*\mathsf{T}}\,p\,,$$
$$x=x\trianglelefteq\xi\trianglerighteq p\Rightarrow x=\xi\,{}^{*\mathsf{F}}\,p\,.$$

Hence, the solutions of the recursion equations $x=p\trianglelefteq\xi\trianglerighteq x$ and $x=x\trianglelefteq\xi\trianglerighteq p$ denoted by the closed terms $\xi\,{}^{*\mathsf{T}}\,p$ and $\xi\,{}^{*\mathsf{F}}\,p$, respectively, are unique solutions of those equations in models for $\mathrm{TA}^*_{\mathrm{sc}}$ in which AIP holds. In Section 3, we will construct models for $\mathrm{TA}_{\mathrm{sc}}$ and $\mathrm{TA}^*_{\mathrm{sc}}$, in which AIP holds.

The properties of the projection operators stated in the following two lemmas are used in coming proofs.

Lemma 2 *For all $p \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$ and $n,m \in \mathbb{N}$, $\pi_n(\pi_m(p)) = \pi_{\min(n,m)}(p)$ is derivable from the axioms of* $\mathrm{TA}^*_{\mathrm{sc}}$ *and axioms* P0–P5.

Proof This is easily proved by induction on $\min(n,m)$, and in the inductive case by induction on the structure of p. □

Lemma 3 *For all $p_1,\ldots,p_m \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$ and $n \in \mathbb{N}$, $\pi_n(\|^{\mathrm{s}}(\langle p_1\rangle \frown \ldots \frown \langle p_m\rangle)) = \|^{\mathrm{s}}(\langle \pi_n(p_1)\rangle \frown \ldots \frown \langle \pi_n(p_m)\rangle)$ is derivable from the axioms of* $\mathrm{TA}^*_{\mathrm{sc}}$ *and axioms* P0–P5.

Proof This is straightforwardly proved by induction on n, and in the inductive case by induction on $\nu(p_1)+\ldots+\nu(p_m)$ and case distinction according to the left-hand side of the axioms for synchronous cooperation. □

The projection operators enable a kind of approximate reduction for each term from $\mathscr{C}$. This is stated in the following proposition.

Proposition 1 *For all $p \in \mathscr{C}$ and $n \in \mathbb{N}$, there exists a term $q \in \mathscr{B}$ such that $\pi_n(p) = q$ is derivable from the axioms of* $\mathrm{TA}^*_{\mathrm{sc}}$ *and axioms* P0–P5.

Proof This is easily proved by induction on n, and in the inductive case by induction on the structure of p. □

Proposition 1 can be generalized from $\mathscr{C}$ to $\mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$, but first we consider a much smaller generalization.

Proposition 2 *For all $p_1,\ldots,p_m \in \mathscr{C}$ and $n \in \mathbb{N}$, there exists a term $q \in \mathscr{B}^0$ such that $\pi_n(\|^{\mathrm{s}}(\langle p_1\rangle \frown \ldots \frown \langle p_m\rangle)) = q$ is derivable from the axioms of* $\mathrm{TA}^*_{\mathrm{sc}}$ *and axioms* P0–P5.

Proof This follows immediately from Lemma 3, Proposition 1 and Corollary 1. □

The following theorem generalizes Proposition 1 from $\mathscr{C}$ to $\mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$.

Theorem 2 *For all $p \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$ and $n \in \mathbb{N}$, there exists a term $q \in \mathscr{B}$ such that $\pi_n(p) = q$ is derivable from the axioms of* $\mathrm{TA}^*_{\mathrm{sc}}$ *and axioms* P0–P5.

Proof The proof follows the same line as the proof of Proposition 1. Here, we have to consider the additional case $p \equiv \|^{\mathrm{s}}(\langle p_1\rangle \frown \ldots \frown \langle p_m\rangle)$, where $p_1,\ldots,p_m \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$. By Lemma 3, $\pi_n(\|^{\mathrm{s}}(\langle p_1\rangle \frown \ldots \frown \langle p_m\rangle)) = \|^{\mathrm{s}}(\langle \pi_n(p_1)\rangle \frown \ldots \frown \langle \pi_n(p_m)\rangle)$. From this and the induction hypothesis, it follows that $\pi_n(\|^{\mathrm{s}}(\langle p_1\rangle \frown \ldots \frown \langle p_m\rangle)) = \|^{\mathrm{s}}(\langle p'_1\rangle \frown \ldots \frown \langle p'_m\rangle)$, for some $p'_1,\ldots,p'_m \in \mathscr{B}$. From this and Proposition 2, it follows that $\pi_n(\|^{\mathrm{s}}(\langle p_1\rangle \frown \ldots \frown \langle p_m\rangle)) = q'$, for some $q' \in \mathscr{B}$. □

The following proposition states a property of synchronous cooperation that cannot be proved without AIP in the presence of conditional action repetition.

Proposition 3 *For all $p \in \mathscr{C}^0$ and $q \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$, $\|^{\mathrm{s}}(p \cdot q) = p \cdot \|^{\mathrm{s}}(q)$ is derivable from the axioms of* $\mathrm{TA}^*_{\mathrm{sc}}$, *axioms* P0–P5 *and* AIP.

Proof We begin by proving that for all $n \in \mathbb{N}$, $\pi_n(\|^{\mathrm{s}}(p \cdot q)) = \pi_n(p \cdot \|^{\mathrm{s}}(q))$. This is easily proved by induction on n and in the inductive case by induction on the structure of p, using Lemma 3. The result then follows by applying AIP. □

This proposition will be used in the proof of Lemma 9.

Table 8 Alphabet axioms

$\alpha(\mathsf{S}) = \emptyset$
$\alpha(\mathsf{D}) = \emptyset$
$\alpha(p \trianglelefteq \xi \trianglerighteq q) = \alpha(p) \cup \alpha(q) \cup \alpha(\xi)$
$\alpha(p \trianglelefteq \mathsf{nt}(r) \trianglerighteq q) = \alpha(p) \cup \alpha(q) \cup \alpha(r)$
$\alpha(p \triangleleft \mathsf{y}_\xi \triangleright q) = \alpha(p) \cup \alpha(q)$
$\alpha(\xi \mathbin{*^b} p) = \alpha(\xi) \cup \alpha(p)$
$\alpha(\|^{\mathsf{s}}(\langle\rangle)) = \emptyset$
$\alpha(\|^{\mathsf{s}}(\langle p_1\rangle \frown \ldots \frown \langle p_m\rangle)) = \alpha(p_1) \cup \ldots \cup \alpha(p_m)$
$\alpha(\delta) = \emptyset$
$\alpha(a_1 \,\&\, \ldots \,\&\, a_n) = \{a_1, \ldots, a_n\}$

2.5 Alphabets

To meet in the need for alphabet extraction, we introduce the unary *alphabet* operator α. Let $p \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$. Then $\alpha(p)$ is the set of all actions from $\mathscr{AA}$ that may be performed by p at some stage. The alphabet axioms are given in Table 8. In this table, $p_1, \ldots, p_m$, p, q and r stand for arbitrary members of $\mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$, ξ stands for an arbitrary member of $\mathscr{CA}_\delta$, $a_1, \ldots, a_n$ stand for arbitrary members of $\mathscr{AA}$, and b stands for an arbitrary member of $\{\mathsf{T}, \mathsf{F}\}$.

The following proposition concerns the alphabet of projections.

Proposition 4 *For all $p \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$ and $n \in \mathbb{N}$, we have $\alpha(\pi_n(p)) \subseteq \alpha(p)$.*

Proof This is straightforwardly proved by induction on n, and in the inductive case by induction on the structure of p. □

The alphabets of threads play a part in the properties of threads that will be given in Section 5.2.

2.6 Structural Operational Semantics of $\mathrm{TA}^*_{\mathrm{sc}}$

We present a structural operational semantics for $\mathrm{TA}^*_{\mathrm{sc}}$. This structural operational semantics is intended to give an operational intuition of the constants and operators of $\mathrm{TA}^*_{\mathrm{sc}}$. We do not construct a model for $\mathrm{TA}^*_{\mathrm{sc}}$ based on the structural operational semantics and an appropriate version of bisimilarity. In Section 3.1, an alternative model for $\mathrm{TA}^*_{\mathrm{sc}}$ is constructed.

In the structural operational semantics, we represent an execution environment by a function $\rho : \mathscr{CA}^* \to (\mathscr{CA} \to \{\mathsf{T}, \mathsf{F}\})$ that satisfies the following conditions:[2]

- if $\alpha \in \mathscr{CA}^*$, $a_1, \ldots, a_{n+1} \in \mathscr{AA}$ are such that $a_1 \,\&\, \ldots \,\&\, a_{n+1} \neq \delta$, and $\alpha' \in \mathrm{perm}(\langle a_1\rangle \frown \ldots \frown \langle a_n\rangle)$, then $\rho(\langle a_1 \,\&\, \ldots \,\&\, a_{n+1}\rangle \frown \alpha) = \rho(\alpha' \frown \langle a_{n+1}\rangle \frown \alpha)$;
- if $\alpha \in \mathscr{CA}^*$, then $\rho(\langle \mathsf{tau}\rangle \frown \alpha) = \rho(\alpha)$;

[2] We write D^* for the set of all finite sequences with elements from set D, and $\mathrm{perm}(\alpha)$ for the set of all permutations of finite sequence α.

Table 9 Transition rules of BTA_δ

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

$$\frac{}{\langle x \trianglelefteq \delta \trianglerighteq y, \rho\rangle\uparrow} \qquad \frac{\rho(\langle\xi\rangle)(\xi) = \mathsf{T}}{\langle x \trianglelefteq \xi \trianglerighteq y, \rho\rangle \xrightarrow{\xi} \langle x, \frac{\partial}{\partial \xi}\rho\rangle} \qquad \frac{\rho(\langle\xi\rangle)(\xi) = \mathsf{F}}{\langle x \trianglelefteq \xi \trianglerighteq y, \rho\rangle \xrightarrow{\xi} \langle y, \frac{\partial}{\partial \xi}\rho\rangle}$$

- if $\alpha \in \mathscr{CA}^*$ and $\xi, \xi' \in \mathscr{CA}$ are such that $\xi \,\&\, \xi' \neq \delta$, then $\rho(\alpha)(\xi \,\&\, \xi') = \rho(\alpha)(\xi')$;
- if $\alpha \in \mathscr{CA}^*$, then $\rho(\alpha)(\mathsf{tau}) = \mathsf{T}$;
- if $\alpha \in \mathscr{CA}^*$ and $\xi, \xi' \in \mathscr{CA}$ are such that $\xi \,\&\, \xi' \neq \delta$, then $\rho(\alpha \frown \langle \xi \,\&\, \xi'\rangle)(\xi) = \rho(\alpha \frown \langle\xi\rangle)(\xi)$ and $\rho(\alpha \frown \langle \xi \,\&\, \xi'\rangle)(\xi') = \rho(\alpha \frown \langle\xi'\rangle)(\xi')$.

We write $\mathscr{E}$ for the set of all those functions. Let $\rho \in \mathscr{E}$, and let $\xi \in \mathscr{CA}$. Then the *derived* execution environment $\frac{\partial}{\partial \xi}\rho$ is defined by $\frac{\partial}{\partial \xi}\rho(\alpha) = \rho(\langle\xi\rangle \frown \alpha)$.

The chosen representation of execution environments is based on the assumption that it depends at any stage only on the history, i.e. the sequence of actions processed before, and the action being processed whether the reply produced is positive or negative. This is a realistic assumption for deterministic execution environments. If the processing of an action amounts to the simultaneous processing of two or more other actions, then the replies produced for each of those actions are considered to be available at completion of the processing as well. For that reason, execution environments cannot simply be represented by functions $\rho : \mathscr{CA}^* \to \{\mathsf{T}, \mathsf{F}\}$.

We write $\mathscr{A}_{\mathsf{nt}}$ for the set $\mathscr{A} \cup \{\mathsf{nt}(p) \mid p \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}\}$.

The following transition relations on closed terms are used in the structural operational semantics of $\mathrm{TA}^*_{\mathrm{sc}}$:

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

These transition relations can be explained as follows:

- $\langle p, \rho\rangle\downarrow$: in execution environment ρ, thread p cannot but terminate successfully;
- $\langle p, \rho\rangle\uparrow$: in execution environment ρ, thread p cannot but become inactive;
- $\langle p, \rho\rangle \xrightarrow{\xi} \langle p', \rho'\rangle$, where $\xi \in \mathscr{A}$: in execution environment ρ, thread p can perform action ξ and after that proceed as thread p' in execution environment ρ';
- $\langle p, \rho\rangle \xrightarrow{\mathsf{nt}(p'')} \langle p', \rho'\rangle$: in execution environment ρ, thread p can fork off thread p'' and after that proceed as thread p' in execution environment ρ'.

The structural operational semantics of $\mathrm{TA}^*_{\mathrm{sc}}$ is described by the transition rules given in Tables 9, 10 and 11. In these tables, $k \geq l > 0$, ξ and ξ' stand for arbitrary actions from $\mathscr{A}$, and ζ_i ($i \in I$) stands for an arbitrary element from $\mathscr{A}_{\mathsf{nt}}$. Moreover, b stands for an arbitrary bijective function from $[1, |I|]$ to I such that, for all $n \in [1, |I|]$, $b(n) \leq b(|I|)$.

Table 10 Additional transition rules for $\mathrm{TA}_{\mathrm{sc}}$

$$\frac{\langle x_1,\rho\rangle\downarrow,\ldots,\langle x_k,\rho\rangle\downarrow}{\langle \|^{\mathsf{s}}(\langle x_1\rangle \frown \ldots \frown \langle x_k\rangle),\rho\rangle\downarrow} \qquad \frac{\langle x_l,\rho\rangle\uparrow}{\langle \|^{\mathsf{s}}(\langle x_1\rangle \frown \ldots \frown \langle x_k\rangle),\rho\rangle\uparrow}$$

$$\frac{\{\langle x_i,\rho\rangle \xrightarrow{\zeta_i} \langle x_i',\rho_i'\rangle \mid i\in I\},\{\langle x_j,\rho\rangle\downarrow \mid j\in J\},\zeta'_{b(1)}\,\&\ldots\&\,\zeta'_{b(|I|)}\neq\delta,\ I\neq\emptyset, I\cap J=\emptyset, I\cup J=[1,k]}{\langle \|^{\mathsf{s}}(\langle x_1\rangle \frown \ldots \frown \langle x_k\rangle),\rho\rangle \xrightarrow{\zeta'_{b(1)}\&\ldots\&\zeta'_{b(|I|)}} \langle \|^{\mathsf{s}}(\alpha'_{b(1)} \frown \ldots \frown \alpha'_{b(|I|)}), \frac{\partial}{\partial\zeta'_{b(1)}\&\ldots\&\zeta'_{b(|I|)}}\rho\rangle}$$

where $\alpha_i' \equiv \langle x_i'\rangle$ and $\zeta_i' = \zeta_i$ if $\zeta_i \in \mathscr{C}\!\mathscr{A}$,
$\alpha_i' \equiv \langle x_i\rangle \frown \langle p\rangle$ and $\zeta_i' = \mathsf{tau}$ if $\zeta_i = \mathsf{nt}(p)$

$$\frac{}{\langle x \trianglelefteq \mathsf{nt}(z) \trianglerighteq y,\rho\rangle \xrightarrow{\mathsf{nt}(z)} \langle x,\rho\rangle}$$

$$\frac{\langle x,\rho\rangle\downarrow}{\langle x \lhd \mathsf{y}_\delta \rhd y,\rho\rangle\downarrow} \qquad \frac{\langle x,\rho\rangle\uparrow}{\langle x \lhd \mathsf{y}_\delta \rhd y,\rho\rangle\uparrow} \qquad \frac{\langle x,\rho\rangle \xrightarrow{\xi'} \langle x',\rho'\rangle}{\langle x \lhd \mathsf{y}_\delta \rhd y,\rho\rangle \xrightarrow{\xi'} \langle x',\rho'\rangle}$$

$$\frac{\langle x,\rho\rangle\downarrow,\rho(\langle\rangle)(\xi)=\mathsf{T}}{\langle x \lhd \mathsf{y}_\xi \rhd y,\rho\rangle\downarrow} \qquad \frac{\langle x,\rho\rangle\uparrow,\rho(\langle\rangle)(\xi)=\mathsf{T}}{\langle x \lhd \mathsf{y}_\xi \rhd y,\rho\rangle\uparrow} \qquad \frac{\langle x,\rho\rangle \xrightarrow{\xi'} \langle x',\rho'\rangle,\rho(\langle\rangle)(\xi)=\mathsf{T}}{\langle x \lhd \mathsf{y}_\xi \rhd y,\rho\rangle \xrightarrow{\xi'} \langle x',\rho'\rangle}$$

$$\frac{\langle y,\rho\rangle\downarrow,\rho(\langle\rangle)(\xi)=\mathsf{F}}{\langle x \lhd \mathsf{y}_\xi \rhd y,\rho\rangle\downarrow} \qquad \frac{\langle y,\rho\rangle\uparrow,\rho(\langle\rangle)(\xi)=\mathsf{F}}{\langle x \lhd \mathsf{y}_\xi \rhd y,\rho\rangle\uparrow} \qquad \frac{\langle y,\rho\rangle \xrightarrow{\xi'} \langle y',\rho'\rangle,\rho(\langle\rangle)(\xi)=\mathsf{F}}{\langle x \lhd \mathsf{y}_\xi \rhd y,\rho\rangle \xrightarrow{\xi'} \langle y',\rho'\rangle}$$

Table 11 Additional transition rules for $\mathrm{TA}^*_{\mathrm{sc}}$

$$\frac{}{\langle \delta \,{}^{*\mathsf{T}}\, x,\rho\rangle\uparrow} \qquad \frac{\rho(\langle\xi\rangle)(\xi)=\mathsf{T}}{\langle \xi \,{}^{*\mathsf{T}}\, x,\rho\rangle \xrightarrow{\xi} \langle x, \frac{\partial}{\partial\xi}\rho\rangle} \qquad \frac{\rho(\langle\xi\rangle)(\xi)=\mathsf{F}}{\langle \xi \,{}^{*\mathsf{T}}\, x,\rho\rangle \xrightarrow{\xi} \langle \xi \,{}^{*\mathsf{T}}\, x, \frac{\partial}{\partial\xi}\rho\rangle}$$

$$\frac{}{\langle \delta \,{}^{*\mathsf{F}}\, x,\rho\rangle\uparrow} \qquad \frac{\rho(\langle\xi\rangle)(\xi)=\mathsf{T}}{\langle \xi \,{}^{*\mathsf{F}}\, x,\rho\rangle \xrightarrow{\xi} \langle \xi \,{}^{*\mathsf{F}}\, x, \frac{\partial}{\partial\xi}\rho\rangle} \qquad \frac{\rho(\langle\xi\rangle)(\xi)=\mathsf{F}}{\langle \xi \,{}^{*\mathsf{F}}\, x,\rho\rangle \xrightarrow{\xi} \langle x, \frac{\partial}{\partial\xi}\rho\rangle}$$

The third transition rule from Table 10 looks more complicated than it actually is. It can be explained as follows: if the threads in a thread vector can be divided into active threads that can make a step by performing an action or forking off a thread and threads that can terminate successfully, and it is possible that all steps concerned are made simultaneously, then the synchronous cooperation of the threads in the thread vector can make all steps concerned simultaneously and after that proceed as the synchronous cooperation of what is left of the active threads in the thread vector, where each thread that forked off a thread gives rise to an additional thread next to it. The threads in the resulting thread vector may also be permuted, with the exception of the thread or threads resulting from the last active thread in the original thread vector. The execution environment changes in accordance with the steps made.

Example 4 Consider the term $\|^{\mathsf{s}}(\langle a \circ (a' \circ \mathsf{S})\rangle \frown \langle \mathsf{nt}(b \circ \mathsf{S}) \circ (b' \circ \mathsf{S})\rangle \frown \langle c \circ (c' \circ \mathsf{S})\rangle)$, where $a,a',b,b',c,c' \in \mathscr{A}\!\mathscr{A}$. Suppose that $a \,\&\, c \neq \delta$. Applying the fourth and fifth

transition rules in Table 9, we obtain:

$$\langle a \circ (a' \circ \mathsf{S}), \rho \rangle \xrightarrow{a} \langle a' \circ \mathsf{S}, \tfrac{\partial}{\partial a}\rho \rangle \;,$$
$$\langle \mathsf{nt}(b \circ \mathsf{S}) \circ (b' \circ \mathsf{S}), \rho \rangle \xrightarrow{\mathsf{nt}(b \circ \mathsf{S})} \langle b' \circ \mathsf{S}, \rho \rangle \;,$$
$$\langle c \circ (c' \circ \mathsf{S}), \rho \rangle \xrightarrow{c} \langle c' \circ \mathsf{S}, \tfrac{\partial}{\partial c}\rho \rangle \;.$$

Next, applying the third transition rule in Table 10, we obtain

$$\langle \|^{\mathsf{s}}(\langle a \circ (a' \circ \mathsf{S}) \rangle \frown \langle \mathsf{nt}(b \circ \mathsf{S}) \circ (b' \circ \mathsf{S}) \rangle \frown \langle c \circ (c' \circ \mathsf{S}) \rangle), \rho \rangle$$
$$\xrightarrow{a\&\mathsf{tau}\&c} \langle \|^{\mathsf{s}}(\langle a' \circ \mathsf{S} \rangle \frown \langle b' \circ \mathsf{S} \rangle \frown \langle b \circ \mathsf{S} \rangle \frown \langle c' \circ \mathsf{S} \rangle), \tfrac{\partial}{\partial a\&\mathsf{tau}\&c}\rho \rangle \;,$$

because $a \,\&\, c \neq \delta$.

Construction of a model for $\mathrm{TA}^*_{\mathrm{sc}}$ based on the structural operational semantics of $\mathrm{TA}^*_{\mathrm{sc}}$ and an appropriate version of bisimilarity is feasible only if that version of bisimilarity is a congruence with respect to the operators of $\mathrm{TA}^*_{\mathrm{sc}}$. To our knowledge, this cannot be established by means of results from the theory of structural operational semantics concerning transition rule formats guaranteeing that some version of bisimilarity is a congruence. It appears that some results from [25, 26] are the nearest obtainable, but there are still difficult issues that must be dealt with. One of those issues is that Theorem 34 from [26] is not applicable for the following reason: in the third transition rule from Table 10, $\rho'_i \neq \frac{\partial}{\partial \zeta'_{b(1)} \& \ldots \& \zeta'_{b(|I|)}}\rho$ for all $i \in I$. We believe that this point does not mean that the version of bisimilarity concerned is not a congruence, but that sufficient conditions for it that are weaker than the ones from the above-mentioned theorem must be found. Another issue is that transition labels containing terms are found in the structural operational semantics of $\mathrm{TA}^*_{\mathrm{sc}}$: this is not covered in [26]. We believe that adaptation on the lines of [25] is possible, but it is not a trivial matter. Exploring all this is considered outside the scope of this paper. Because a projective limit model for $\mathrm{TA}^*_{\mathrm{sc}}$ is most appropriate to the justification of the induction principle that is introduced in Section 5.1, we decided to construct a projective limit model instead of a model based on the structural operational semantics.

3 Projective Limit Model for $\mathrm{TA}^*_{\mathrm{sc}}$

In this section, we construct the projective limit model for $\mathrm{TA}^*_{\mathrm{sc}}$. First, we construct the projective limit model for $\mathrm{TA}_{\mathrm{sc}}$. Next, we make the domain of this model into a metric space and show that every guarded recursion equation has a unique solution in this domain using Banach's fixed point theorem. Finally, we expand the projective limit model for $\mathrm{TA}_{\mathrm{sc}}$ to a model for $\mathrm{TA}^*_{\mathrm{sc}}$ using this uniqueness result.

3.1 Projective Limit Model for $\mathrm{TA}_{\mathrm{sc}}$

We construct the projective limit model for $\mathrm{TA}_{\mathrm{sc}}$. In this model infinite threads are represented by infinite sequences of finite approximations.

To express definitions more concisely, the interpretations of the constants and operators from the signature of $\mathrm{TA}_{\mathrm{sc}}$ in the initial model for $\mathrm{TA}_{\mathrm{sc}}$ and the projective limit model for $\mathrm{TA}_{\mathrm{sc}}$ are denoted by the constants and operators themselves. The ambiguity thus introduced could be obviated by decorating the symbols, with different decorations for different models, when they are used to denote their interpretation in a model. However, in this paper, it is always immediately clear from the context how the symbols are used. Moreover, we believe that the decorations are more often than not distracting. Therefore, we leave it to the reader to mentally decorate the symbols wherever appropriate.

The projective limit construction is known as the inverse limit construction in domain theory, the theory underlying the approach of denotational semantics for programming languages (see e.g. [27]). In process algebra, this construction has been applied for the first time by Bergstra and Klop [4].

We will write A_ω for the domain of the initial model for $\mathrm{TA}_{\mathrm{sc}}$. A_ω consists of the equivalence classes of basic terms with respect to the equivalence induced by the axioms of $\mathrm{TA}_{\mathrm{sc}}$. In other words, modulo equivalence, A_ω is $\mathscr{B}$. Henceforth, we will identify basic terms with their equivalence class.

Each element of A_ω represents a finite thread, i.e. a thread of which the length of the sequences of actions that it can perform is bounded. Below, we will construct a model that covers infinite threads as well. In preparation for that, we define for all n a function that cuts off finite threads from A_ω after performing a sequence of actions of length n.

For all $n \in \mathbb{N}$, we have the *projection* operation $\pi_n : A_\omega \to A_\omega$, inductively defined by

$$\begin{aligned}
&\pi_0(p) = \mathsf{D}\,,\\
&\pi_{n+1}(\mathsf{S}) = \mathsf{S}\,,\\
&\pi_{n+1}(\mathsf{D}) = \mathsf{D}\,,\\
&\pi_{n+1}(p \trianglelefteq \xi \trianglerighteq q) = \pi_n(p) \trianglelefteq \xi \trianglerighteq \pi_n(q)\,,\\
&\pi_{n+1}(p \trianglelefteq \mathsf{nt}(r) \trianglerighteq q) = \pi_n(p) \trianglelefteq \mathsf{nt}(\pi_n(r)) \trianglerighteq \pi_n(q)\,,\\
&\pi_{n+1}(p \triangleleft \mathsf{y}_\xi \triangleright q) = \pi_{n+1}(p) \triangleleft \mathsf{y}_\xi \triangleright \pi_{n+1}(q)\,.
\end{aligned}$$

For $p \in A_\omega$, $\pi_n(p)$ is called the n-th projection of p. It can be thought of as an approximation of p. If $\pi_n(p) \neq p$, then $\pi_{n+1}(p)$ can be thought of as the closest better approximation of p. If $\pi_n(p) = p$, then $\pi_{n+1}(p) = p$ as well. For all $n \in \mathbb{N}$, we will write A_n for $\{\pi_n(p) \mid p \in A_\omega\}$.

The semantic equations given above to define the projection operations have the same shape as the axioms for the projection operators introduced in Section 2.4. We will come back to the definition of the projection operations at the end of Section 3.3.

The properties of the projection operations stated in the following two lemmas will be used frequently in the sequel.

Lemma 4 *For all $p \in A_\omega$ and $n, m \in \mathbb{N}$, we have $\pi_n(\pi_m(p)) = \pi_{\min(n,m)}(p)$.*

Proof This is easily proved by induction on the structure of p. □

Lemma 5 *For all $p_1, \ldots, p_m \in A_\omega$ and $n \in \mathbb{N}$, we have $\pi_n(\|^{\mathsf{s}}(\langle p_1\rangle \frown \ldots \frown \langle p_m\rangle)) = \|^{\mathsf{s}}(\langle \pi_n(p_1)\rangle \frown \ldots \frown \langle \pi_n(p_m)\rangle)$.*

Proof This is straightforwardly proved by induction on $\nu(p_1)+\ldots+\nu(p_m)$ and case distinction according to the left-hand sides of the axioms for synchronous cooperation. □

In the projective limit model, which covers finite and infinite threads, threads are represented by *projective sequences*, i.e. infinite sequences $(p_n)_{n\in\mathbb{N}}$ of elements of A_ω such that $p_n \in A_n$ and $p_n = \pi_n(p_{n+1})$ for all $n \in \mathbb{N}$. In other words, a projective sequence is a sequence of which successive components are successive projections of the same thread. The idea is that any infinite thread is fully characterized by the infinite sequence of all its finite approximations. We will write A^∞ for $\{(p_n)_{n\in\mathbb{N}} \mid \bigwedge_{n\in\mathbb{N}}(p_n \in A_n \wedge p_n = \pi_n(p_{n+1}))\}$.

The *projective limit model* for $\mathrm{TA}_{\mathrm{sc}}$ consists of the following:

- the set A^∞, the domain of the projective limit model;
- an element of A^∞ for each constant of $\mathrm{TA}_{\mathrm{sc}}$;
- an operation on A^∞ for each operator of $\mathrm{TA}_{\mathrm{sc}}$;

where those elements of A^∞ and operations on A^∞ are defined as follows:

$$\begin{aligned}
&\mathsf{S} && = (\pi_n(\mathsf{S}))_{n\in\mathbb{N}}\,,\\
&\mathsf{D} && = (\pi_n(\mathsf{D}))_{n\in\mathbb{N}}\,,\\
&(p_n)_{n\in\mathbb{N}} \trianglelefteq \xi \trianglerighteq (q_n)_{n\in\mathbb{N}} && = (\pi_n(p_n \trianglelefteq \xi \trianglerighteq q_n))_{n\in\mathbb{N}}\,,\\
&(p_n)_{n\in\mathbb{N}} \trianglelefteq \mathsf{nt}((r_n)_{n\in\mathbb{N}}) \trianglerighteq (q_n)_{n\in\mathbb{N}} && = (\pi_n(p_n \trianglelefteq \mathsf{nt}(r_n) \trianglerighteq q_n))_{n\in\mathbb{N}}\,,\\
&(p_n)_{n\in\mathbb{N}} \triangleleft \mathsf{y}_\xi \triangleright (q_n)_{n\in\mathbb{N}} && = (\pi_n(p_n \triangleleft \mathsf{y}_\xi \triangleright q_n))_{n\in\mathbb{N}}\,,\\
&\|^{\mathsf{s}}(\langle (p_{1n})_{n\in\mathbb{N}}\rangle \frown \ldots \frown \langle (p_{mn})_{n\in\mathbb{N}}\rangle) && = (\pi_n(\|^{\mathsf{s}}(\langle p_{1n}\rangle \frown \ldots \frown \langle p_{mn}\rangle)))_{n\in\mathbb{N}}\,.
\end{aligned}$$

Using Lemmas 4 and 5, we easily prove for $(p_n)_{n\in\mathbb{N}}, (q_n)_{n\in\mathbb{N}}, (r_n)_{n\in\mathbb{N}} \in A^\infty$ and $(p_{1n})_{n\in\mathbb{N}}, \ldots, (p_{mn})_{n\in\mathbb{N}} \in A^\infty$:

- $\pi_n(\pi_{n+1}(p_{n+1} \trianglelefteq \xi \trianglerighteq q_{n+1})) = \pi_n(p_n \trianglelefteq \xi \trianglerighteq q_n)$;
- $\pi_n(\pi_{n+1}(p_{n+1} \trianglelefteq \mathsf{nt}(r_{n+1}) \trianglerighteq q_{n+1})) = \pi_n(p_n \trianglelefteq \mathsf{nt}(r_n) \trianglerighteq q_n)$;
- $\pi_n(\pi_{n+1}(p_{n+1} \triangleleft \mathsf{y}_\xi \triangleright q_{n+1})) = \pi_n(p_n \triangleleft \mathsf{y}_\xi \triangleright q_n)$;
- $\pi_n(\pi_{n+1}(\|^{\mathsf{s}}(\langle p_{1n+1}\rangle \frown \ldots \frown \langle p_{mn+1}\rangle))) = \pi_n(\|^{\mathsf{s}}(\langle p_{1n}\rangle \frown \ldots \frown \langle p_{mn}\rangle))$.

From this and the definition of A_n, it follows immediately that the operations defined above are well-defined, i.e. they always yield elements of A^∞.

The initial model can be embedded in a natural way in the projective limit model: each $p \in A_\omega$ corresponds to $(\pi_n(p))_{n\in\mathbb{N}} \in A^\infty$. We extend projection to an operation on A^∞ by defining $\pi_m((p_n)_{n\in\mathbb{N}}) = (p'_n)_{n\in\mathbb{N}}$, where $p'_n = p_n$ if $n < m$ and $p'_n = p_m$ if $n \geq m$. That is, $\pi_m((p_n)_{n\in\mathbb{N}})$ is p_m embedded in A^∞ as described above. Henceforth, we will identify elements of A_ω with their embedding in A^∞ where elements of A^∞ are concerned.

For each $\xi \in \mathscr{A}_\delta$, the operations corresponding to the conditional action repetition operators $\xi\,{}^{*\mathsf{T}}\_$ and $\xi\,{}^{*\mathsf{F}}\_$ of $\mathrm{TA}^*_{\mathrm{sc}}$ can be thought of as solutions in A^∞ of parametrized equations suggested by axioms CAR1 and CAR2. That is, for all $p \in A^\infty$, $\xi^{*\mathsf{T}} p$ is thought of as a solution in A^∞ of the equation $x = p \trianglelefteq \xi \trianglerighteq x$ and $\xi^{*\mathsf{F}} p$ is thought of as a solution in A^∞ of the equation $x = x \trianglelefteq \xi \trianglerighteq p$. The question is whether these equations have unique solutions in A^∞. This question can be answered in the affirmative by mean of a result that will be established in Section 3.3.

3.2 Metric Space Structure for Projective Limit Model

In Section 3.3, we will introduce the notion of guarded recursion equation and show that every guarded recursion equation has a unique solution in A^∞. Following [22] to some extent, we make A^∞ into a metric space to establish the uniqueness of solutions of guarded recursion equations using Banach's fixed point theorem.

Supplementary, in Appendix A, we make A^∞ into a complete partial ordered set and show, using Tarski's fixed point theorem, that every recursion equation has a least solution in A^∞ with respect to the partial order relation concerned.

We remark that metric spaces have also been applied in concurrency theory by de Bakker and others to solve domain equations for process domains [2] and to establish uniqueness results for recursion equations [1].

In the remainder of this subsection, as well as in Section 3.3, we assume known the notions of metric space, completion of a metric space, dense subset in a metric space, continuous function on a metric space, limit in a metric space and contracting function on a metric space, and Banach's fixed point theorem. The definitions of the above-mentioned notions concerning metric spaces and Banach's fixed point theorem can, for example, be found in [17]. In this paper, we will consider only ultrametric spaces. A metric space (M,d) is an *ultrametric space* if for all $p,p',p'' \in M$, $d(p,p') \leq \max\{d(p,p''),d(p'',p')\}$.

We define a distance function $d : A^\infty \times A^\infty \to \mathbb{R}$ by

$$d(p,p') = 2^{-\min\{n \in \mathbb{N} \mid \pi_n(p) \neq \pi_n(p')\}} \quad \text{if } p \neq p' \,,$$
$$d(p,p') = 0 \quad \text{if } p = p' \,.$$

It is easy to verify that (A^∞,d) is a *metric space*. The following theorem summarizes the basic properties of this metric space.

Theorem 3

1. *(A^∞,d) is an ultrametric space;*
2. *(A^∞,d) is the metric completion of the metric space (A_ω,d'), where d' is the restriction of d to A_ω;*
3. *A_ω is dense in A^∞;*
4. *the operations $\pi_n : A^\infty \to A_n$ are continuous;*
5. *for all $p \in A^\infty$ and $n \in \mathbb{N}$, $d(\pi_n(p),p) < 2^{-n}$, hence $\lim_{n\to\infty} \pi_n(p) = p$.*

Proof These properties are general properties of metric spaces constructed in the way pursued here. Proofs of the first three properties can be found in [28]. A proof of the fourth property can be found in [18]. The fifth property is proved as follows. It follows from Lemma 4, by passing to the limit and using that the projection operations are continuous and A_ω is dense in A^∞, that $\pi_n(\pi_m(p)) = \pi_{\min(n,m)}(p)$ for $p \in A^\infty$ as well. Hence, $\min\{m \in \mathbb{N} \mid \pi_m(\pi_n(p)) \neq \pi_m(p)\} > n$, and consequently $d(\pi_n(p),p) < 2^{-n}$. □

The basic properties given above are used in coming proofs.

The properties of the projection operations stated in the following lemma will be used in the proof of Theorem 4 given below.

Lemma 6 *For all $p_1,\ldots,p_m \in A^\infty$ and $n \in \mathbb{N}$:*

$$\pi_n(p_1 \trianglelefteq \xi \trianglerighteq p_2) = \pi_n(\pi_n(p_1) \trianglelefteq \xi \trianglerighteq \pi_n(p_2)) ,$$
$$\pi_n(p_1 \trianglelefteq \mathsf{nt}(p_3) \trianglerighteq p_2) = \pi_n(\pi_n(p_1) \trianglelefteq \mathsf{nt}(\pi_n(p_3)) \trianglerighteq \pi_n(p_2)) ,$$
$$\pi_n(p_1 \triangleleft \mathsf{y}_\xi \triangleright p_2) = \pi_n(\pi_n(p_1) \triangleleft \mathsf{y}_\xi \triangleright \pi_n(p_2)) ,$$
$$\pi_n(\|^{\mathsf{s}}(\langle p_1\rangle \frown \ldots \frown \langle p_m\rangle)) = \pi_n(\|^{\mathsf{s}}(\langle \pi_n(p_1)\rangle \frown \ldots \frown \langle \pi_n(p_m)\rangle)) .$$

Proof It is enough to prove these equalities for $p_1,\ldots,p_m \in A_\omega$. The lemma will then follow by passing to the limit and using that π_n is continuous and A_ω is dense in A^∞. For $p_1,\ldots,p_m \in A_\omega$, the first three equalities follow immediately from Lemma 4 and the definition of π_n and the fourth equality follows immediately from Lemmas 4 and 5. □

In the terminology of metric topology, the following theorem states that all operations in the projective limit model for $\mathrm{TA}_{\mathrm{sc}}$ are non-expansive. This implies that they are continuous, with respect to the metric topology induced by d, in all arguments.

Theorem 4 *For all $p_1,\ldots,p_m,p'_1,\ldots,p'_m \in A^\infty$:*

$$d(p_1 \trianglelefteq \xi \trianglerighteq p_2, p'_1 \trianglelefteq \xi \trianglerighteq p'_2) \leq \max(d(p_1,p'_1), d(p_2,p'_2)) ,$$
$$d(p_1 \trianglelefteq \mathsf{nt}(p_3) \trianglerighteq p_2, p'_1 \trianglelefteq \mathsf{nt}(p'_3) \trianglerighteq p'_2) \leq \max(d(p_1,p'_1), d(p_2,p'_2), d(p_3,p'_3)) ,$$
$$d(p_1 \triangleleft \mathsf{y}_\xi \triangleright p_2, p'_1 \triangleleft \mathsf{y}_\xi \triangleright p'_2) \leq \max(d(p_1,p'_1), d(p_2,p'_2)) ,$$
$$d(\|^{\mathsf{s}}(\langle p_1\rangle \frown \ldots \frown \langle p_m\rangle), \|^{\mathsf{s}}(\langle p'_1\rangle \frown \ldots \frown \langle p'_m\rangle)) \leq \max(d(p_1,p'_1),\ldots,d(p_m,p'_m)) .$$

Proof Let $k_i = \min\{n \in \mathbb{N} \mid \pi_n(p_i) \neq \pi_n(p'_i)\}$ for $i = 1,2$, and let $k = \min(k_1,k_2)$. Then for all $n \in \mathbb{N}$, $n < k$ iff $\pi_n(p_1) = \pi_n(p'_1)$ and $\pi_n(p_2) = \pi_n(p'_2)$. From this and the first equality from Lemma 6, it follows immediately that $\pi_{k-1}(p_1 \trianglelefteq \xi \trianglerighteq p_2) = \pi_{k-1}(p'_1 \trianglelefteq \xi \trianglerighteq p'_2)$. Hence, $k \leq \min\{n \in \mathbb{N} \mid \pi_n(p_1 \trianglelefteq \xi \trianglerighteq p_2) \neq \pi_n(p'_1 \trianglelefteq \xi \trianglerighteq p'_2)\}$, which completes the proof for the first inequality. The proofs for the other inequalities go analogously. □

3.3 Guarded Recursion Equations

We introduce the notion of guarded recursion equation and show that each guarded recursion equation has a unique solution in A^∞. Before we introduce the notion of guarded recursion equation, we introduce several other notions relevant to the issue of unique solutions of recursion equations.

We assume that there is a fixed but arbitrary set of variables $\mathscr{X}$. We will write $\mathscr{T}_P$, where $P \subseteq A^\infty$, for the set of all terms over the signature of $\mathrm{TA}_{\mathrm{sc}}$ with parameters from P; and $\mathscr{T}_P^X$, where $P \subseteq A^\infty$ and $X \subseteq \mathscr{X}$, for the set of all terms from $\mathscr{T}_P$ in which no other variables than the ones in X have free occurrences.[3] The interpretation function $[\![\_]\!] : \mathscr{T}_P \to ((\mathscr{X} \to A^\infty) \to A^\infty)$ of terms with parameters from

[3] A term with parameters is a term in which elements of the domain of a model are used as constants naming themselves. For a justification of this mix-up of syntax and semantics in case only one model is under consideration, see e.g. [19].

$P \subseteq A^\infty$ is defined as usual for terms without parameters, but with the additional defining equation $[\![p]\!](\rho) = p$ for parameters p.

Let $x_1,\ldots,x_n \in \mathscr{X}$, let $X \subseteq \{x_1,\ldots,x_n\}$, let $P \subseteq A^\infty$, and let $t \in \mathscr{T}_P^X$. Moreover, let $\rho : \mathscr{X} \to A^\infty$. Then the *interpretation of t with respect to $x_1,\ldots,x_n$*, written $[\![t]\!]^{x_1,\ldots,x_n}$, is the unique function $\phi : A^{\infty n} \to A^\infty$ such that for all $p_1,\ldots,p_n \in A^\infty$, $\phi(p_1,\ldots,p_n) = [\![t]\!](\rho \oplus [x_1 \mapsto p_1] \oplus \ldots \oplus [x_n \mapsto p_n])$.

The interpretation of t with respect to $x_1,\ldots,x_n$ is well-defined because it is independent of the choice of ρ.

An m-ary operation ϕ on A^∞ is a *guarded* operation if for all $p_1,\ldots,p_m$, $p'_1,\ldots,p'_m \in A^\infty$ and $n \in \mathbb{N}$:

$$\begin{aligned}&\pi_n(p_1) = \pi_n(p'_1) \wedge \ldots \wedge \pi_n(p_m) = \pi_n(p'_m)\\ &\quad \Rightarrow \pi_{n+1}(\phi(p_1,\ldots,p_m)) = \pi_{n+1}(\phi(p'_1,\ldots,p'_m))\,.\end{aligned}$$

We say that ϕ is an *unguarded* operation if ϕ is not a guarded operation.

The notion of guarded operation, which originates from [28], supersedes the notion of guard used in [22].

The notion of guarded operation is defined without reference to metric properties. However, being a guarded operation coincides with having a metric property that is highly relevant to the issue of unique solutions of recursion equations: an operation on A^∞ is a guarded operation iff it is contracting. This is stated in the following lemma.

Lemma 7 *An m-ary operation ϕ on A^∞ is a guarded operation iff for all $p_1,\ldots,p_m,p'_1,\ldots,p'_m \in A^\infty$:*

$$d(\phi(p_1,\ldots,p_m),\phi(p'_1,\ldots,p'_m)) \leq \tfrac{1}{2}\cdot\max(d(p_1,p'_1),\ldots,d(p_m,p'_m))\,.$$

Proof Let $k_i = \min\{n \in \mathbb{N} \mid \pi_n(p_i) \neq \pi_n(p'_i)\}$ for $i = 1,\ldots,m$, and let $k = \min\{k_1,\ldots,k_m\}$. Then for all $n \in \mathbb{N}$, $n < k$ iff $\pi_n(p_1) = \pi_n(p'_1)$ and $\ldots$ and $\pi_n(p_m) = \pi_n(p'_m)$. From this, the definition of a guarded operation and the definition of π_0, it follows immediately that ϕ is a guarded operation iff for all $n < k+1$, $\pi_n(\phi(p_1,\ldots,p_m)) = \pi_n(\phi(p'_1,\ldots,p'_m))$. Hence, ϕ is a guarded operation iff $k+1 \leq \min\{n \in \mathbb{N} \mid \pi_n(\phi(p_1,\ldots,p_m)) \neq \pi_n(\phi(p'_1,\ldots,p'_m))\}$, which completes the proof. □

The notion of guarded term defined below is suggested by the fact, stated in Lemma 7 above, that an operation on A^∞ is a guarded operation iff it is contracting. The only guarded operations, and consequently contracting operations, in the projective limit model for $\mathrm{TA}_{\mathrm{sc}}$ are the non-forking and forking postconditional composition operations. Based upon this, we define the notion of guarded term as follows.

Let $P \subseteq A^\infty$. Then the set $\mathscr{G}_P$ of *guarded* terms with parameters from P is inductively defined as follows:

- if $p \in P$, then $p \in \mathscr{G}_P$;
- $\mathsf{S}, \mathsf{D} \in \mathscr{G}_P$;
- if $\xi \in \mathscr{A}_\delta$ and $t_1, t_2 \in \mathscr{T}_P$, then $t_1 \trianglelefteq \xi \trianglerighteq t_2 \in \mathscr{G}_P$;
- if $t_1, t_2, t_3 \in \mathscr{T}_P$, then $t_1 \trianglelefteq \mathsf{nt}(t_3) \trianglerighteq t_2 \in \mathscr{G}_P$;

– if $\xi \in \mathscr{A}_\delta$ and $t_1, t_2 \in \mathscr{G}_P$, then $t_1 \trianglelefteq \mathsf{y}_\xi \trianglerighteq t_2 \in \mathscr{G}_P$;
– if $t_1, \ldots, t_l \in \mathscr{G}_P$, then $\|^{\mathrm{s}}(\langle t_1 \rangle \frown \ldots \frown \langle t_l \rangle) \in \mathscr{G}_P$.

The following lemma states that guarded terms represent operations on A^∞ that are contracting.

Lemma 8 *Let $x_1, \ldots, x_n \in \mathscr{X}$, let $X \subseteq \{x_1, \ldots, x_n\}$, let $P \subseteq A^\infty$, and let $t \in \mathscr{T}_P^X$. Then $t \in \mathscr{G}_P$ only if for all $p_1, \ldots, p_n, p'_1, \ldots, p'_n \in A^\infty$:*

$$d([\![t]\!]^{x_1,\ldots,x_n}(p_1,\ldots,p_n), [\![t]\!]^{x_1,\ldots,x_n}(p'_1,\ldots,p'_n)) \\ \leq \tfrac{1}{2} \cdot \max\{d(p_1,p'_1),\ldots,d(p_n,p'_n)\} \,.$$

Proof This is easily proved by induction on the structure of t using Theorem 4, Lemma 7, and the fact that the non-forking and forking postconditional composition operations are guarded operations. □

A *recursion equation* is an equation $x = t$, where $x \in \mathscr{X}$ and $t \in \mathscr{T}_P^{\{x\}}$ for some $P \subseteq A^\infty$. A recursion equation $x = t$ is a *guarded* recursion equation if $t \in \mathscr{G}_P$ for some $P \subseteq A^\infty$. Let $x = t$ be a recursion equation. Then $p \in A^\infty$ is a *solution* of $x = t$ if $[\![t]\!]^x(p) = p$.

Every guarded recursion equation has a unique solution in the projective limit model for $\mathrm{TA}_{\mathrm{sc}}$. This is stated in the following theorem.

Theorem 5 *Let $x \in \mathscr{X}$, let $P \subseteq A^\infty$, and let $t \in \mathscr{T}_P^{\{x\}}$ be such that $t \in \mathscr{G}_P$. Then the guarded recursion equation $x = t$ has a unique solution in the projective limit model for* $\mathrm{TA}_{\mathrm{sc}}$.

Proof We have from Theorem 3 that (A^∞, d) is a complete metric space and from Lemma 8 that $[\![t]\!]^x$ is contracting. From this, we conclude by Banach's fixed point theorem that there exists a unique $p \in A^\infty$ such that $[\![t]\!]^x(p) = p$. Hence, the guarded recursion equation $x = t$ has a unique solution. □

For completeness, we mention how the unique solution of a guarded recursion equation $x = t$ can be constructed. Define the iterates ϕ^n of a unary operation ϕ on A^∞ by induction on n as follows: $\phi^0(p) = p$ and $\phi^{n+1}(p) = \phi(\phi^n(p))$. The unique solution of $x = t$ in A^∞ is $(\pi_n(([\![t]\!]^x)^n(\mathsf{D})))_{n\in\mathbb{N}}$.

Example 5 The equation $x = x \trianglelefteq \xi \trianglerighteq \mathsf{S}$, where $\xi \in \mathscr{A}$, is a guarded recursion equation. The unique solution of this recursion equation is the projective sequence $(p_n)_{n\in\mathbb{N}}$, where:

$$\begin{aligned} p_0 &= \mathsf{D}\,, \\ p_1 &= \mathsf{D} \trianglelefteq \xi \trianglerighteq \mathsf{D}\,, \\ p_2 &= (\mathsf{D} \trianglelefteq \xi \trianglerighteq \mathsf{D}) \trianglelefteq \xi \trianglerighteq \mathsf{S}\,, \\ p_3 &= ((\mathsf{D} \trianglelefteq \xi \trianglerighteq \mathsf{D}) \trianglelefteq \xi \trianglerighteq \mathsf{S}) \trianglelefteq \xi \trianglerighteq \mathsf{S}\,, \\ &\vdots \end{aligned}$$

Theorem 5 is a considerable generalization of a result on unique solutions of recursion equations given in [30]. That result can be rephased as follows: every guarded recursion equation with a right-hand side that contains no other constants and operators than S, D and $\_ \trianglelefteq \xi \trianglerighteq \_$ (for $\xi \in \mathscr{A}_\delta$) has a unique solution in the projective limit model for BTA_δ.

The projection operations and the distance function as defined in this paper match well with our intuitive ideas about finite approximations of threads and closeness of threads, respectively. The suitability of the definitions given in this paper is supported by the fact that guarded operations coincide with contracting operations. However, it is not at all clear whether adaptations of the definitions are feasible and will lead to different uniqueness results.

3.4 Expansion of Projective Limit Model for $\mathrm{TA}_{\mathrm{sc}}$ to Model for $\mathrm{TA}^*_{\mathrm{sc}}$

The expansion of the projective limit model for $\mathrm{TA}_{\mathrm{sc}}$ to a model for $\mathrm{TA}^*_{\mathrm{sc}}$ rests heavily upon Sections 3.2 and 3.3.

The projective limit model for $\mathrm{TA}^*_{\mathrm{sc}}$ is the expansion of the projective limit model for $\mathrm{TA}_{\mathrm{sc}}$ with:

– an operation for each conditional action repetition operator;

where those additional operations are defined as follows:

$$\xi {}^{*\mathsf{T}} p \text{ is the unique solution of } x = p \trianglelefteq \xi \trianglerighteq x \,,$$
$$\xi {}^{*\mathsf{F}} p \text{ is the unique solution of } x = x \trianglelefteq \xi \trianglerighteq p \,.$$

Because the equations $x = p \trianglelefteq \xi \trianglerighteq x$ and $x = x \trianglelefteq \xi \trianglerighteq p$ are guarded recursion equations, they have unique solutions in A^∞ by Theorem 5. Moreover, those solutions are the intended ones: axioms CAR1 and CAR2 hold in the model expanded in this way.

The definitions of the operations for conditional action repetition clarify why we decided on considering terms with parameters in Section 3.3. We would have been able to carry on with terms without parameters, but that would have been a needless burden.

Notice that Theorem 5 justifies an extension of $\mathrm{TA}_{\mathrm{sc}}$ or $\mathrm{TA}^*_{\mathrm{sc}}$ with guarded recursion. We will not work out the details of such an extension in this paper.

4 Threads and Maurer Machines

In this section, we introduce Maurer machines and add application of a thread to a Maurer machine from one of its state to the thread algebra developed so far. We also introduce the notion of computation in the resulting setting. However, we start with a brief review of Maurer computers.

4.1 Maurer Computers

Maurer computers are computers as defined by Maurer in [23].

A *Maurer computer* C consists of the following components:

- a non-empty set M;
- a set B with $\mathrm{card}(B) \geq 2$;
- a set $\mathscr{S}$ of functions $S : M \to B$;
- a set $\mathscr{O}$ of functions $O : \mathscr{S} \to \mathscr{S}$;

and satisfies the following conditions:

- if $S_1, S_2 \in \mathscr{S}$, $M' \subseteq M$ and $S_3 : M \to B$ is such that $S_3(x) = S_1(x)$ if $x \in M'$ and $S_3(x) = S_2(x)$ if $x \notin M'$, then $S_3 \in \mathscr{S}$;
- if $S_1, S_2 \in \mathscr{S}$, then the set $\{x \in M \mid S_1(x) \neq S_2(x)\}$ is finite.

M is called the *memory*, B is called the *base set*, the members of $\mathscr{S}$ are called the *states*, and the members of $\mathscr{O}$ are called the *operations*. It is obvious that the first condition is satisfied if C is *complete*, i.e. if $\mathscr{S}$ is the set of all functions $S : M \to B$, and that the second condition is satisfied if C is *finite*, i.e. if M and B are finite sets.

In [23], operations are called instructions. In the current paper, the term operation is used because of the confusion that would otherwise arise with the instructions of which program algebra programs are made up.

The memory of a Maurer computer consists of memory elements which have as contents an element from the base set of the Maurer computer. The contents of all memory elements together make up a state of the Maurer computer. The operations of the Maurer computer transform states in certain ways and thus change the contents of certain memory elements. We return to the conditions on the states of a Maurer computer after the introduction of the input region and output region of an operation.

Let $(M, B, \mathscr{S}, \mathscr{O})$ be a Maurer computer, and let $O : \mathscr{S} \to \mathscr{S}$. Then the *input region* of O, written $IR(O)$, and the *output region* of O, written $OR(O)$, are the subsets of M defined as follows:[4]

$$IR(O) = \{u \in M \mid \exists S_1, S_2 \in \mathscr{S} \bullet (\forall w \in M \setminus \{u\} \bullet S_1(w) = S_2(w) \wedge \exists v \in OR(O) \bullet O(S_1)(v) \neq O(S_2)(v))\} ,$$

$$OR(O) = \{u \in M \mid \exists S \in \mathscr{S} \bullet S(u) \neq O(S)(u)\} .$$

$OR(O)$ is the set of all memory elements that are possibly affected by O; and $IR(O)$ is the set of all memory elements that possibly affect elements of $OR(O)$ under O.

Let $(M, B, \mathscr{S}, \mathscr{O})$ be a Maurer computer, let $S_1, S_2 \in \mathscr{S}$, and let $O \in \mathscr{O}$. Then $S_1 \restriction IR(O) = S_2 \restriction IR(O)$ implies $O(S_1) \restriction OR(O) = O(S_2) \restriction OR(O)$.[5] The conditions

[4] The following precedence conventions are used in logical formulas. Operators bind stronger than predicate symbols, and predicate symbols bind stronger than logical connectives and quantifiers. Moreover, $\neg$ binds stronger than $\wedge$ and $\vee$, and $\wedge$ and $\vee$ bind stronger than $\Rightarrow$ and $\Leftrightarrow$. Quantifiers are given the smallest possible scope.

[5] In this paper, we use the notation $f \restriction D$, where f is a function and $D \subseteq \mathrm{dom}(f)$, for the function g with $\mathrm{dom}(g) = D$ such that for all $d \in \mathrm{dom}(g)$, $g(d) = f(d)$.

on the states of a Maurer computer are necessary for this desirable property to hold.

Let $(M,B,\mathscr{S},\mathscr{O})$ be a Maurer computer, let $O \in \mathscr{O}$, let $M' \subseteq OR(O)$, and let $M'' \subseteq IR(O)$. Then the *region affecting M' under O*, written $RA(M',O)$, and the *region affected by M'' under O*, written $AR(M'',O)$, are the subsets of M defined as follows:

$$RA(M',O) = \{u \in IR(O) \mid AR(\{u\},O) \cap M' \neq \emptyset\} ,$$

$$AR(M'',O) = \{u \in OR(O) \mid \exists S_1,S_2 \in \mathscr{S} \bullet (\forall w \in IR(O) \setminus M'' \bullet S_1(w) = S_2(w) \wedge O(S_1)(u) \neq O(S_2)(u))\} .$$

$AR(M'',O)$ is the set of all elements of $OR(O)$ that are possibly affected by the elements of M'' under O; and $RA(M',O)$ is the set of all elements of $IR(O)$ that possibly affect elements of M' under O.

In [23], Maurer gives many results about the relation between the input region and output region of operations, the composition of operations, the decomposition of operations and the existence of operations. In [8], we summarize the main results given in [23]. Recently, a revised and expanded version of [23], which includes all the proofs, has appeared in [24].

4.2 Applying Threads to Maurer Machines

We introduce Maurer machines and add for a fixed but arbitrary Maurer machine a binary *apply* operator $\_ \bullet \_$ to $\mathrm{TA}^*_{\mathrm{sc}}$, resulting in $\mathrm{TA}^{*\bullet}_{\mathrm{sc}}$. This operator is related to the apply operators introduced in [15].

Below, we expand Maurer computers $(M,B,\mathscr{S},\mathscr{O})$ with a set A, a function $[\![\_]\!] : A \to (\mathscr{O} \times M)$ and a relation $C \subseteq A \times A$ to obtain Maurer machines. For each $a \in A$, we will write O_a and m_a for the unique $O \in \mathscr{O}$ and $m \in M$, respectively, such that $[\![a]\!] = (O,m)$.

A *Maurer machine* is a tuple $H = (M,B,\mathscr{S},\mathscr{O},A,[\![\_]\!],C)$, where:

- $(M,B,\mathscr{S},\mathscr{O})$ is a Maurer computer;
- A is a set with $\mathsf{tau} \in A$ and $\delta \notin A$;
- $[\![\_]\!] : A \to (\mathscr{O} \times M)$ is such that:
  - for all $a \in A$: $\forall S \in \mathscr{S} \bullet S(\mathrm{m}_a) \in \{\mathsf{T},\mathsf{F}\}$;
  - $\forall S \in \mathscr{S} \bullet (\mathrm{O}_{\mathsf{tau}}(S) = S \wedge S(\mathrm{m}_{\mathsf{tau}}) = \mathsf{T})$;
- $C \subseteq A \times A$ is such that for all $a,b \in A$:

$$\begin{aligned} & C(a,b) \Rightarrow \\ & \forall S \in \mathscr{S} \bullet (\mathrm{O}_a(\mathrm{O}_b(S)) = \mathrm{O}_b(\mathrm{O}_a(S)) \wedge \\ & \qquad \mathrm{O}_a(\mathrm{O}_b(S))(\mathrm{m}_b) = \mathrm{O}_b(S)(\mathrm{m}_b) \wedge \\ & \qquad \mathrm{O}_b(\mathrm{O}_a(S))(\mathrm{m}_a) = \mathrm{O}_a(S)(\mathrm{m}_a)) . \end{aligned}$$

The members of A are called the *atomic actions* of H, and $[\![\_]\!]$ is called the *atomic action interpretation function* of H. C is called the *atomic action concurrency relation* of H.

Let $H = (M, B, \mathscr{S}, \mathscr{O}, A, [\![\_]\!], C)$ be a Maurer machine. A, $[\![\_]\!]$ and C constitute the interface between the Maurer machine and its environment. The interface can be explained as follows:

- $a \in A$ means that H is capable of processing atomic action a;
- for $a \in A$, $[\![a]\!] = (O, m)$ means that:
  - the processing of atomic action a by H amounts to performing operation O,
  - after that the reply produced by H is contained in memory element m;
- for $a, b \in A$, $C(a, b)$ means that the atomic actions a and b can be processed concurrently.

The condition imposed on C sees to it that atomic actions a and b can be processed concurrently only if in the case where a and b are processed by H one after another:

- the ultimate effect on the contents of memory elements never depends on the order in which the actions are processed;
- the contents of the memory cell containing the reply produced in processing the first action remains unchanged when the other action is processed.

This condition concerns aspects of real computers which are relevant to program parallelization, but from which the well-known models for computers abstract.

In [8–10], the interface of a Maurer machine did not include an atomic action concurrency relation. Its inclusion is needed to be able to determine the correctness of any program parallelization statically.

Let $H = (M, B, \mathscr{S}, \mathscr{O}, A, [\![\_]\!], C)$ be a Maurer machine. A condition that is stronger than the condition imposed on C can be expressed in terms of the input regions and output regions of operations:

$$\begin{aligned}
&C(a,b) \Rightarrow \\
&OR(\mathrm{O}_a) \cap IR(\mathrm{O}_b) = IR(\mathrm{O}_a) \cap OR(\mathrm{O}_b) = OR(\mathrm{O}_a) \cap OR(\mathrm{O}_b) = \emptyset \wedge \\
&\mathrm{m}_a \notin OR(\mathrm{O}_b) \wedge \mathrm{m}_b \notin OR(\mathrm{O}_a)
\end{aligned}$$

for all $a, b \in A$. This stronger condition may be useful in establishing that the intended atomic action concurrency relation of a Maurer machine under construction is really the atomic action concurrency relation of the Maurer machine according to the definition of the notion of Maurer machine given above.

In $\mathrm{TA}^{*\bullet}_{\mathrm{sc}}$, it is assumed that a fixed but arbitrary Maurer machine $H = (M, B, \mathscr{S}, \mathscr{O}, A, [\![\_]\!], C)$ has been given that satisfies the following conditions:

- $\mathscr{AA} = A$;
- for all $a_1, \ldots, a_n \in A$: $a_1 \,\&\, \ldots \,\&\, a_n \neq \delta$ iff $\bigwedge_{1 \leq i < n} \bigwedge_{i < j \leq n} C(a_i, a_j)$;
- for all $a_1, \ldots, a_m \in A$ and $a'_1, \ldots, a'_n \in A$: $a_1 \,\&\, \ldots \,\&\, a_m = a'_1 \,\&\, \ldots \,\&\, a'_n$ iff $\mathrm{O}_{a_m}(\ldots \mathrm{O}_{a_1}(S) \ldots) = \mathrm{O}_{a'_n}(\ldots \mathrm{O}_{a'_1}(S) \ldots)$ for all $S \in \mathscr{S}$;
- for all $a, b \in A$ with $a \neq b$: $\forall S \in \mathscr{S} \bullet \mathrm{O}_a(S)(\mathrm{m}_b) = S(\mathrm{m}_b)$.

Wherever this assumption is made, the notations O_a and m_a introduced above will be used. The following notations will also be used. Let $\xi = a_1 \,\&\, \ldots \,\&\, a_n$

Table 12 Axioms for apply

$x \bullet \uparrow = \uparrow$
$\mathsf{S} \bullet S = S$
$\mathsf{D} \bullet S = \uparrow$
$(x \trianglelefteq \xi \trianglerighteq y) \bullet S = x \bullet \mathrm{O}_\xi(S)$ if $\mathrm{O}_\xi(S)(\mathrm{m}_\xi) = \mathsf{T}$
$(x \trianglelefteq \xi \trianglerighteq y) \bullet S = y \bullet \mathrm{O}_\xi(S)$ if $\mathrm{O}_\xi(S)(\mathrm{m}_\xi) = \mathsf{F}$
$(x \trianglelefteq \mathsf{nt}(z) \trianglerighteq y) \bullet S = \uparrow$
$(x \triangleleft \mathrm{y}_\xi \triangleright y) \bullet S = x \bullet S$ if $S(\mathrm{m}_\xi) = \mathsf{T}$
$(x \triangleleft \mathrm{y}_\xi \triangleright y) \bullet S = y \bullet S$ if $S(\mathrm{m}_\xi) = \mathsf{F}$

Table 13 Rule for divergence

$\bigwedge_{n \geq 0} \pi_n(x) \bullet S = \uparrow \Rightarrow x \bullet S = \uparrow$

with $a_1, \ldots, a_n \in A$ and $\xi \neq \delta$. Then we write O_ξ for the unique $O \in \mathscr{O}$ such that $O(S) = \mathrm{O}_{a_n}(\ldots \mathrm{O}_{a_1}(S)\ldots)$ for all $S \in \mathscr{S}$, and we write m_ξ for m_{a_n}.

The apply operator $\_ \bullet \_$ allows for threads to transform states of the Maurer machine H by means of its operations. Such state transformations produce either a state of the associated Maurer machine or the *undefined state* $\uparrow$. It is assumed that $\uparrow$ is not a state of any Maurer machine. We extend function restriction to $\uparrow$ by stipulating that $\uparrow \restriction M = \uparrow$ for any set M. The first operand of the apply operator must be a term from $\mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$ and its second operand must be a state from $\mathscr{S} \cup \{\uparrow\}$. Let $p \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$, and let $S \in \mathscr{S}$. Then $p \bullet S$ is the state from $\mathscr{S}$ that results if all actions from $\mathscr{CA}$ performed by thread p are processed by the Maurer machine H from initial state S. The processing of an action ξ from $\mathscr{CA}$ by H amounts to a state change according to the operation O_ξ. In the resulting state the reply produced by H is contained in memory element m_ξ. If p is S, then there will be no state change. If p is D, then the result is $\uparrow$.

The axioms for apply are given in Tables 12 and 13. In these tables, ξ stands for an arbitrary member of $\mathscr{CA}$ and S stands for an arbitrary member of $\mathscr{S}$. The reason for the equation $(x \trianglelefteq \mathsf{nt}(z) \trianglerighteq y) \bullet S = \uparrow$ is that no actions will become available for processing by the Maurer machine because thread forking is carried into effect only if it is put in the context of synchronous cooperation.

Let $p \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$ and $S \in \mathscr{S}$. Then p *converges* from S if there exists an $n \in \mathbb{N}$ such that $\pi_n(p) \bullet S \neq \uparrow$. We say that p *diverges* from S if it does not converge from S. The rule for divergence from Table 13 can be read as follows: if x diverges from S, then $x \bullet S$ equals $\uparrow$.

4.3 Computations

We introduce the notion of computation and related notions in the current setting.

The *step* relation $\_ \vdash \_ \subseteq (\mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}} \times \mathscr{S}) \times (\mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}} \times \mathscr{S})$ is inductively defined as follows:

- if $p = \mathsf{tau} \circ p'$, then $(p,S) \vdash (p',S)$;
- if $\xi \neq \delta$, $\mathrm{O}_\xi(S)(\mathrm{m}_\xi) = \mathsf{T}$ and $p = p' \trianglelefteq \xi \trianglerighteq p''$, then $(p,S) \vdash (p', \mathrm{O}_\xi(S))$;

- if $\xi \neq \delta$, $\mathrm{O}_\xi(S)(\mathrm{m}_\xi) = \mathsf{F}$ and $p = p' \trianglelefteq \xi \trianglerighteq p''$, then $(p,S) \vdash (p'', \mathrm{O}_\xi(S))$;
- if $\xi \neq \delta$, $\mathrm{O}_\xi(S)(\mathrm{m}_\xi) = \mathsf{T}$, $p = q \trianglelefteq \mathrm{y}_\xi \trianglerighteq r$, and $(q,S) \vdash (q',S')$, then $(p,S) \vdash (q',S')$;
- if $\xi \neq \delta$, $\mathrm{O}_\xi(S)(\mathrm{m}_\xi) = \mathsf{F}$, $p = q \trianglelefteq \mathrm{y}_\xi \trianglerighteq r$, and $(r,S) \vdash (r',S')$, then $(p,S) \vdash (r',S')$.

A *full* path in $\_ \vdash \_$ is one of the following:

- a finite path $\langle (p_0,S_0), \ldots, (p_n,S_n) \rangle$ in $\_ \vdash \_$ such that there does not exist a $(p_{n+1},S_{n+1}) \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}} \times \mathscr{S}$ with $(p_n,S_n) \vdash (p_{n+1},S_{n+1})$;
- an infinite path $\langle (p_0,S_0), (p_1,S_1), \ldots \rangle$ in $\_ \vdash \_$.

Let $p \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$, and let $S \in \mathscr{S}$. Then the *full path* of (p,S) is the unique full path in $\_ \vdash \_$ from (p,S). The *computation* of (p,S) is the full path of (p,S) if p converges from S and undefined otherwise.

Let $p \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$ and $S \in \mathscr{S}$ be such that p converges from S. Then we write $\|(p,S)\|$ for the length of the computation of (p,S).

It is easy to see that $(p_0,S_0) \vdash (p_1,S_1)$ only if $p_0 \bullet S_0 = p_1 \bullet S_1$ and that $\langle (p_0,S_0), \ldots, (p_n,S_n) \rangle$ is the computation of (p_0,S_0) only if $p_n = \mathsf{S}$ and $S_n = p_0 \bullet S_0$. It is also easy to see that, if p_0 converges from S_0, $\|(p_0,S_0)\|$ is the least $n \in \mathbb{N}$ such that $\pi_n(p_0) \bullet S_0 \neq \uparrow$.

Notice that, because $(p \trianglelefteq \mathsf{nt}(r) \trianglerighteq q) \bullet S = \uparrow$ for all $p,q,r \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$ and $S \in \mathscr{S}$, there are no computations for threads involving thread forking.

Program instructions whose processing takes one step can be looked upon as atomic actions of a Maurer machine. A program instruction whose processing takes more than one step can be handled by means of split-action prefixing (see Section 2.3) with two atomic actions, say a and b, using some memory element as a counter:

- in the case where the instruction takes n steps ($n > 1$):
  - operation O_a sets a counter to $n-1$ and sets m_a to F,
  - operation O_b decrements the counter by one and sets m_b to T if the value of the decremented counter is zero and to F otherwise;
- in the case where the instruction takes n to m steps ($m > n > 1$):
  - operation O_a sets a counter to a value in the interval $[n-1, m-1]$ depending upon the contents of certain memory elements and sets m_a to F,
  - operation O_b decrements the counter by one and sets m_b to T if the value of the decremented counter is zero and to F otherwise.

Both cases can occur, for example, with load instructions – the second case due to the possibility of cache misses. In the second case, the value to which the counter is set depends on the contents of memory elements that are related to the origin of the varying number of steps. For example, a varying number of steps due to the possibility of cache misses means that the value to which the counter is set depends on the contents of memory elements that model the mechanism of the cache. For each individual computer architecture, it is reasonable to assume that a lower bound and upper bound on the number of steps taken by each instruction can be given.

Table 14 Defining formula for state transformer equivalence

$x \approx y \Leftrightarrow \forall S \in \mathscr{S} \bullet (x \bullet S = y \bullet S)$

5 Threads as State Transformers

In this section, we introduce the notion of state transformer equivalence of threads and present some state transformer properties of threads.

5.1 State Transformer Equivalence

We introduce state transformer equivalence of threads. This equivalence identifies threads if they are the same as transformers of the states of the Maurer machine H. An interesting point of state transformer equivalence is the following: if p and q are state transformer equivalent, then the computations of (p,S) and (q,S) have the same final state, but they may have different lengths.

State transformer equivalence, written $\approx$, is defined by the formula given in Table 14. The following proposition states some basic properties of state transformer equivalence.

Proposition 5 *For all* $\xi, \xi' \in \mathscr{CA}_\delta$*:*

$$\xi \circ \mathsf{D} \approx \mathsf{D} \,, \tag{1}$$

$$\mathsf{tau} \circ x \approx x \,, \tag{2}$$

$$\xi \mathbin{\&} \xi' \neq \delta \Rightarrow x \trianglelefteq \xi \mathbin{\&} \xi' \trianglerighteq y \approx \xi \circ (x \trianglelefteq \xi' \trianglerighteq y) \,, \tag{3}$$

$$\begin{aligned} &\xi \mathbin{\&} \xi' \neq \delta \\ &\quad \Rightarrow (x \trianglelefteq \xi' \trianglerighteq y) \trianglelefteq \xi \trianglerighteq (z \trianglelefteq \xi' \trianglerighteq w) \approx (x \trianglelefteq \xi \trianglerighteq z) \trianglelefteq \xi' \trianglerighteq (y \trianglelefteq \xi \trianglerighteq w) \,, \end{aligned} \tag{4}$$

$$(x \triangleleft \mathsf{y}_{\xi'} \triangleright y) \trianglelefteq \xi \trianglerighteq (z \triangleleft \mathsf{y}_{\xi'} \triangleright w) \approx (x \trianglelefteq \xi \trianglerighteq z) \triangleleft \mathsf{y}_{\xi'} \triangleright (y \trianglelefteq \xi \trianglerighteq w) \,, \tag{5}$$

$$(x \triangleleft \mathsf{y}_{\xi'} \triangleright y) \triangleleft \mathsf{y}_{\xi} \triangleright (z \triangleleft \mathsf{y}_{\xi'} \triangleright w) \approx (x \triangleleft \mathsf{y}_{\xi} \triangleright z) \triangleleft \mathsf{y}_{\xi'} \triangleright (y \triangleleft \mathsf{y}_{\xi} \triangleright w) \,. \tag{6}$$

Proof These properties follow easily from the defining formula for state transformer equivalence, the defining equations for the apply operator, the definition of a Maurer machine, and the assumptions made about the Maurer machine H. □

The laws of state transformer equivalence given above are used in coming proofs.

All threads represented by closed terms over the signature of $\mathrm{TA}_{\mathrm{sc}}$ are finite threads. The length of the sequences of actions that a finite thread can perform is bounded. This has the effect that, if two threads represented by closed terms over the signature of $\mathrm{TA}_{\mathrm{sc}}$ are state transformer equivalent, then this can be proved from the axioms of $\mathrm{TA}_{\mathrm{sc}}$ and the defining equations of the apply operator. However, all threads represented by closed terms over the signature of $\mathrm{TA}^*_{\mathrm{sc}}$ other than closed terms over the signature of $\mathrm{TA}_{\mathrm{sc}}$ are infinite threads. As a result of that, the axioms of $\mathrm{TA}^*_{\mathrm{sc}}$ and the defining equations of the apply operator are not sufficient to prove state transformer equivalence.

This calls for a proof rule to deal with infinite threads. A complication that must be dealt with is the following: different threads can effect the same state transformation by performing different sequences of actions. This leads us to the

Table 15 Defining formula for state transformer inclusion

$x \mathrel{\underset{\sim}{\sqsubseteq}} y \Leftrightarrow \forall S \in \mathscr{S} \bullet (x \bullet S \neq \uparrow \Rightarrow x \bullet S = y \bullet S)$

Table 16 State transformer inclusion principle

$\forall n \in \mathbb{N} \bullet \exists m \in \mathbb{N} \bullet \pi_n(x) \mathrel{\underset{\sim}{\sqsubseteq}} \pi_m(y) \Rightarrow x \mathrel{\underset{\sim}{\sqsubseteq}} y$

introduction of state transformer inclusion of threads. Intuitively, one thread includes another thread as state transformer if each state transformation that can be effected by the former thread can be effected by the latter thread as well.

State transformer inclusion, written $\mathrel{\underset{\sim}{\sqsubseteq}}$, is defined by the formula given in Table 15. The following proposition states basic properties of state transformer inclusion.

Proposition 6 *For all $\xi \in \mathscr{CA}_\delta$:*

$$x \mathrel{\underset{\sim}{\sqsubseteq}} x\,, \tag{1}$$
$$x \mathrel{\underset{\sim}{\sqsubseteq}} y \wedge y \mathrel{\underset{\sim}{\sqsubseteq}} z \Rightarrow x \mathrel{\underset{\sim}{\sqsubseteq}} z\,, \tag{2}$$
$$x \mathrel{\underset{\sim}{\sqsubseteq}} y \wedge y \mathrel{\underset{\sim}{\sqsubseteq}} x \Leftrightarrow x \approx y\,, \tag{3}$$
$$x \mathrel{\underset{\sim}{\sqsubseteq}} z \wedge y \mathrel{\underset{\sim}{\sqsubseteq}} u \Rightarrow x \trianglelefteq \xi \trianglerighteq y \mathrel{\underset{\sim}{\sqsubseteq}} z \trianglelefteq \xi \trianglerighteq u\,, \tag{4}$$
$$x \mathrel{\underset{\sim}{\sqsubseteq}} z \wedge y \mathrel{\underset{\sim}{\sqsubseteq}} u \Rightarrow x \triangleleft \mathsf{y}_\xi \triangleright y \mathrel{\underset{\sim}{\sqsubseteq}} z \triangleleft \mathsf{y}_\xi \triangleright u\,. \tag{5}$$

Proof These properties follow easily from the defining formula for state transformer inclusion, the defining formula for state transformer equivalence, and the defining equations for the apply operator. □

Now we are ready to introduce a rule to prove that one infinite thread includes another infinite thread as state transformer. The rule concerned, called the *state transformer inclusion principle*, is given in Table 16. To prove that two infinite threads p and q are state transformer equivalent, the intended approach is to prove $p \mathrel{\underset{\sim}{\sqsubseteq}} q$ and $q \mathrel{\underset{\sim}{\sqsubseteq}} p$ using the state transformer inclusion principle. That is sufficient by Property 3 from Proposition 6.

The following proposition states some basic properties of state transformer equivalence that can be proved following this approach.

Proposition 7 *For all $\xi, \xi' \in \mathscr{CA}_\delta$ and $b \in \{\mathsf{T}, \mathsf{F}\}$:*

$$\xi \mathbin{\&} \xi' \neq \delta \Rightarrow (\xi'^{*b}\, x) \trianglelefteq \xi \trianglerighteq (\xi'^{*b}\, y) \approx \xi'^{*b}\, (x \trianglelefteq \xi \trianglerighteq y)\,,$$
$$\xi \mathbin{\&} \xi' \neq \delta \Rightarrow \xi^{*b}\, (\xi'^{*b}\, x) \approx \xi'^{*b}\, (\xi^{*b}\, x)\,.$$

Proof Assume that $\xi \mathbin{\&} \xi' \neq \delta$. Then it is easily proved by induction on n, using Property 4 from Proposition 5, that $\pi_n((\xi'^{*b}\, x) \trianglelefteq \xi \trianglerighteq (\xi'^{*b}\, y)) \approx \pi_n(\xi'^{*b}\, (x \trianglelefteq \xi \trianglerighteq y))$ for all $n \in \mathbb{N}$. From this and Property 3 from Proposition 6, the first property follows immediately by the state transformer inclusion principle. The proof for the second property goes similarly, and makes use of the first property. □

Table 17 Backwards state transformer inclusion principle

$x \mathrel{\underline{\sqsubseteq}} y \Rightarrow \forall n \in \mathbb{N} \bullet \exists m \in \mathbb{N} \bullet \pi_n(x) \mathrel{\underline{\sqsubseteq}} \pi_m(y)$

We have the following corollary from Property 4 from Proposition 5 and Proposition 7.

Corollary 2 *For all* $\xi, \zeta, \xi', \zeta' \in \mathscr{CA}_\delta$*:*

$$\begin{aligned} &\xi \,\&\, \xi' \neq \delta \wedge \xi \,\&\, \zeta' \neq \delta \wedge \zeta \,\&\, \xi' \neq \delta \wedge \zeta \,\&\, \zeta' \neq \delta \\ &\quad \Rightarrow \xi / \zeta \circ (\xi' / \zeta' \circ x) \approx \xi' / \zeta' \circ (\xi / \zeta \circ x) \,. \end{aligned}$$

The following proposition states a useful property of state transformer inclusion that can be proved by means of the state transformer inclusion principle.

Proposition 8 *For all* $p \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$ *and* $n \in \mathbb{N}$, $\pi_n(p) \mathrel{\underline{\sqsubseteq}} \pi_{n+1}(p)$.

Proof Take $n, n' \in \mathbb{N}$. If $n' \leq n$, then $\pi_{n'}(\pi_n(p)) = \pi_{n'}(\pi_{n+1}(p))$ by Lemma 2. If $n' > n$, then $\pi_{n'}(\pi_n(p)) = \pi_n(\pi_{n+1}(p))$ by Lemma 2. This means that for all $n' \in \mathbb{N}$ there exists an $m' \in \mathbb{N}$ such that $\pi_{n'}(\pi_n(p)) = \pi_{m'}(\pi_{n+1}(p))$. Because $x = y$ implies $x \approx y$, it follows immediately by the state transformer inclusion principle that $\pi_n(p) \mathrel{\underline{\sqsubseteq}} \pi_{n+1}(p)$. □

We also introduce the state transformer inclusion principle in the reverse direction, called the *backwards state transformer inclusion principle*. It is given in Table 17.

The following proposition states a basic property of state transformer inclusion that can be proved using the forward and backward state transformer inclusion principles.

Proposition 9 *For all* $p, q \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$, $\xi \in \mathscr{CA}_\delta$ *and* $b \in \{\mathsf{T}, \mathsf{F}\}$*:*

$$p \mathrel{\underline{\sqsubseteq}} q \Rightarrow \xi \mathbin{{}^{*b}} p \mathrel{\underline{\sqsubseteq}} \xi \mathbin{{}^{*b}} q \,.$$

Proof Assume that for all $n \in \mathbb{N}$, there exists an $m \in \mathbb{N}$ such that $\pi_n(p) \mathrel{\underline{\sqsubseteq}} \pi_m(q)$. Then it is easily proved by induction on n, using Property 4 from Proposition 6 and Proposition 8, that for all $n \in \mathbb{N}$, there exists an $m \in \mathbb{N}$ such that $\pi_n(\xi \mathbin{{}^{*b}} p) \mathrel{\underline{\sqsubseteq}} \pi_m(\xi \mathbin{{}^{*b}} q)$. From this and the forward and backward state transformer inclusion principles, it follows that $p \mathrel{\underline{\sqsubseteq}} q \Rightarrow \xi \mathbin{{}^{*b}} p \mathrel{\underline{\sqsubseteq}} \xi \mathbin{{}^{*b}} q$. □

In Appendix A, we introduce behavioural approximation of threads and relate it to state transformer inclusion.

As a preparation to the expansion of the projective limit model for $\mathrm{TA}^*_{\mathrm{sc}}$ with relations for the predicate symbols $\mathrel{\underline{\sqsubseteq}}$ and $\approx$, we introduce a *state transformer extraction* function $\mathrm{sttrf} : A^\infty \to \mathscr{S} \times \mathscr{S}$. This function is defined as follows:

$$\mathrm{sttrf}((p_n)_{n \in \mathbb{N}}) = \bigcup_{n \in \mathbb{N}} \{(S, S') \in \mathscr{S} \times \mathscr{S} \mid p_n \bullet S = S'\} \,.$$

The relations $\underset{\sim}{\sqsubset}$ and $\approx$ on A^∞ associated with the predicate symbols $\underset{\sim}{\sqsubset}$ and $\approx$, respectively, are defined as follows ($p, q \in A^\infty$):

$$p \underset{\sim}{\sqsubset} q \Leftrightarrow \mathrm{sttrf}(p) \subseteq \mathrm{sttrf}(q) \,,$$

$$p \approx q \Leftrightarrow \mathrm{sttrf}(p) = \mathrm{sttrf}(q) \,.$$

It is easy to verify that the formulas in Tables 14–17 are sound with respect to the expansion of the projective limit model for $\mathrm{TA}^*_{\mathrm{sc}}$ defined above.

5.2 State Transformer Properties of Threads

We present some state transformer properties of threads which can be useful when investigating program parallelization. The notation $p \cdot q$, which is mainly used in this subsection, was introduced at the end of Section 2.3.

The following proposition concerns the preservation of state transformer inclusion.

Proposition 10 *Let $p \in \mathscr{C}^0$ and $q, q' \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$. Then $q \underset{\sim}{\sqsubset} q'$ implies $p \cdot q \underset{\sim}{\sqsubset} p \cdot q'$.*

Proof This is easily proved by induction on the structure of p, using Propositions 6 and 9. □

The following proposition concerns re-ordering of threads.

Proposition 11 *Let $p, q \in \mathscr{C}^0$ be such that $a \,\&\, a' \neq \delta$ for all $a \in \alpha(p)$ and $a' \in \alpha(q)$. Then $p \cdot q \approx q \cdot p$.*

Proof This is proved by induction on the structure of p and in the cases $p \equiv p' \trianglelefteq \xi \trianglerighteq p''$, $p \equiv p' \lhd \mathrm{y}_\xi \rhd p''$, $p \equiv \xi \mathbin{*^{\mathsf{T}}} p'$, and $p \equiv \xi \mathbin{*^{\mathsf{F}}} p'$ by induction on the structure of q, using Propositions 5, 6, 7 and 9. The proof is straightforward given the properties stated in those propositions. □

The following proposition concerns parallelization of threads.

Proposition 12 *Let $p, q \in \mathscr{C}^0$ be such that $a \,\&\, a' \neq \delta$ for all $a \in \alpha(p)$ and $a' \in \alpha(q)$. Then $p \cdot q \approx \|^{\mathrm{s}}(\langle p \rangle \frown \langle q \rangle)$.*

Proof This is proved by induction on the structure of p and in the cases $p \equiv p' \trianglelefteq \xi \trianglerighteq p''$, $p \equiv p' \lhd \mathrm{y}_\xi \rhd p''$, $p \equiv \xi \mathbin{*^{\mathsf{T}}} p'$, and $p \equiv \xi \mathbin{*^{\mathsf{F}}} p'$ by case distinction on the structure of q, using Propositions 5, 6 and 11. The proof is tedious, but straightforward given the properties stated in those propositions. We outline the

case where $p \equiv p' \trianglelefteq \xi \trianglerighteq p''$ and $q \equiv \zeta \mathbin{*^{\mathsf{T}}} q'$:

$$
\begin{aligned}
&(p' \trianglelefteq \xi \trianglerighteq p'') \cdot (\zeta \mathbin{*^{\mathsf{T}}} q') \\
\approx\ &(p' \cdot (\zeta \mathbin{*^{\mathsf{T}}} q')) \trianglelefteq \xi \trianglerighteq (p'' \cdot (\zeta \mathbin{*^{\mathsf{T}}} q')) \\
\approx\ &((\zeta \mathbin{*^{\mathsf{T}}} q') \cdot p') \trianglelefteq \xi \trianglerighteq ((\zeta \mathbin{*^{\mathsf{T}}} q') \cdot p'') \\
\approx\ &((q' \trianglelefteq \zeta \trianglerighteq (\zeta \mathbin{*^{\mathsf{T}}} q')) \cdot p') \trianglelefteq \xi \trianglerighteq ((q' \trianglelefteq \zeta \trianglerighteq (\zeta \mathbin{*^{\mathsf{T}}} q')) \cdot p'') \\
\approx\ &((q' \cdot p') \trianglelefteq \zeta \trianglerighteq ((\zeta \mathbin{*^{\mathsf{T}}} q') \cdot p')) \trianglelefteq \xi \trianglerighteq ((q' \cdot p'') \trianglelefteq \zeta \trianglerighteq ((\zeta \mathbin{*^{\mathsf{T}}} q') \cdot p'')) \\
\approx\ &((p' \cdot q') \trianglelefteq \zeta \trianglerighteq (p' \cdot (\zeta \mathbin{*^{\mathsf{T}}} q'))) \trianglelefteq \xi \trianglerighteq ((p'' \cdot q') \trianglelefteq \zeta \trianglerighteq (p'' \cdot (\zeta \mathbin{*^{\mathsf{T}}} q'))) \\
\approx\ &\xi \circ ((\zeta \circ ((p' \cdot q') \lhd \mathsf{y}_\zeta \rhd (p' \cdot (\zeta \mathbin{*^{\mathsf{T}}} q')))) \lhd \mathsf{y}_\xi \rhd \\
&\qquad (\zeta \circ ((p'' \cdot q') \lhd \mathsf{y}_\zeta \rhd (p'' \cdot (\zeta \mathbin{*^{\mathsf{T}}} q'))))) \\
\approx\ &\xi \circ (\zeta \circ (((p' \cdot q') \lhd \mathsf{y}_\zeta \rhd (p' \cdot (\zeta \mathbin{*^{\mathsf{T}}} q'))) \lhd \mathsf{y}_\xi \rhd \\
&\qquad\qquad ((p'' \cdot q') \lhd \mathsf{y}_\zeta \rhd (p'' \cdot (\zeta \mathbin{*^{\mathsf{T}}} q'))))) \\
\approx\ &\xi \mathbin{\&} \zeta \circ (((p' \cdot q') \lhd \mathsf{y}_\zeta \rhd (p' \cdot (\zeta \mathbin{*^{\mathsf{T}}} q'))) \lhd \mathsf{y}_\xi \rhd \\
&\qquad\qquad ((p'' \cdot q') \lhd \mathsf{y}_\zeta \rhd (p'' \cdot (\zeta \mathbin{*^{\mathsf{T}}} q')))) \\
\approx\ &\xi \mathbin{\&} \zeta \circ ((\|^{\mathsf{s}}(\langle p' \rangle \frown \langle q' \rangle) \lhd \mathsf{y}_\zeta \rhd \|^{\mathsf{s}}(\langle p' \rangle \frown \langle \zeta \mathbin{*^{\mathsf{T}}} q' \rangle)) \lhd \mathsf{y}_\xi \rhd \\
&\qquad\qquad (\|^{\mathsf{s}}(\langle p'' \rangle \frown \langle q' \rangle) \lhd \mathsf{y}_\zeta \rhd \|^{\mathsf{s}}(\langle p'' \rangle \frown \langle \zeta \mathbin{*^{\mathsf{T}}} q' \rangle))) \\
\approx\ &\xi \mathbin{\&} \zeta \circ \|^{\mathsf{s}}(\langle p' \lhd \mathsf{y}_\xi \rhd p'' \rangle \frown \langle q' \lhd \mathsf{y}_\zeta \rhd (\zeta \mathbin{*^{\mathsf{T}}} q') \rangle) \\
\approx\ &\|^{\mathsf{s}}(\langle p' \trianglelefteq \xi \trianglerighteq p'' \rangle \frown \langle q' \trianglelefteq \zeta \trianglerighteq (\zeta \mathbin{*^{\mathsf{T}}} q') \rangle) \\
\approx\ &\|^{\mathsf{s}}(\langle p' \trianglelefteq \xi \trianglerighteq p'' \rangle \frown \langle \zeta \mathbin{*^{\mathsf{T}}} q' \rangle) \,.
\end{aligned}
$$

□

Like Proposition 10, the following proposition concerns the preservation of state transformer equivalence.

Proposition 13 *Let $p, q, q' \in \mathscr{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$ be such that $a \mathbin{\&} a' \neq \delta$ for all $a \in \alpha(p)$ and $a' \in \alpha(q) \cup \alpha(q')$. Then $\|^{\mathsf{s}}(\langle q \rangle) \approx \|^{\mathsf{s}}(\langle q' \rangle)$ implies $\|^{\mathsf{s}}(\langle p \rangle \frown \langle q \rangle) \approx \|^{\mathsf{s}}(\langle p \rangle \frown \langle q' \rangle)$.*

Proof Let $n, m \in \mathbb{N}$ be such that $n \leq m$. Then $\pi_n(p) \mathrel{\underline{\sqsubseteq}} \pi_m(p)$ by Proposition 8. From this, Theorem 2, and Propositions 2, 4, 10, 11 and 12, it follows that $\|^{\mathsf{s}}(\langle \pi_n(q) \rangle) \mathrel{\underline{\sqsubseteq}} \|^{\mathsf{s}}(\langle \pi_m(q') \rangle)$ implies $\|^{\mathsf{s}}(\langle \pi_n(p) \rangle \frown \langle \|^{\mathsf{s}}(\langle \pi_n(q) \rangle) \rangle) \mathrel{\underline{\sqsubseteq}} \|^{\mathsf{s}}(\langle \pi_m(p) \rangle \frown \langle \|^{\mathsf{s}}(\langle \pi_m(q') \rangle) \rangle)$. From this and Lemmas 1 and 3, it follows that $\pi_n(\|^{\mathsf{s}}(\langle q \rangle)) \mathrel{\underline{\sqsubseteq}} \pi_m(\|^{\mathsf{s}}(\langle q' \rangle))$ implies $\pi_n(\|^{\mathsf{s}}(\langle p \rangle \frown \langle q \rangle)) \mathrel{\underline{\sqsubseteq}} \pi_m(\|^{\mathsf{s}}(\langle p \rangle \frown \langle q' \rangle))$. From this, Proposition 8 and the forward and backward state transformer inclusion principles, it follows that $\|^{\mathsf{s}}(\langle q \rangle) \mathrel{\underline{\sqsubseteq}} \|^{\mathsf{s}}(\langle q' \rangle)$ implies $\|^{\mathsf{s}}(\langle p \rangle \frown \langle q \rangle) \mathrel{\underline{\sqsubseteq}} \|^{\mathsf{s}}(\langle p \rangle \frown \langle q' \rangle)$. It follows by symmetry that also $\|^{\mathsf{s}}(\langle q' \rangle) \mathrel{\underline{\sqsubseteq}} \|^{\mathsf{s}}(\langle q \rangle)$ implies $\|^{\mathsf{s}}(\langle p \rangle \frown \langle q' \rangle) \mathrel{\underline{\sqsubseteq}} \|^{\mathsf{s}}(\langle p \rangle \frown \langle q \rangle)$. Hence, $\|^{\mathsf{s}}(\langle q \rangle) \approx \|^{\mathsf{s}}(\langle q' \rangle)$ implies $\|^{\mathsf{s}}(\langle p \rangle \frown \langle q \rangle) \approx \|^{\mathsf{s}}(\langle p \rangle \frown \langle q' \rangle)$. □

6 Programs

In this section, we introduce the classes of programs that are considered in our study of program parallelization in Section 7. All programs concerned are considered closed terms of a program algebra, which is introduced in this section as

well. In this program algebra, the behaviour of a program under execution is taken for a thread. For a clear picture of the threads that are involved, we start with introducing the classes of threads that correspond to the classes of programs that are considered in the study of program parallelization.

6.1 Relevant Classes of Threads

The classes of programs that are considered in the study of program parallelization are in essence sequences of instructions in which test, jump and fork instructions do not occur and sequences of instructions in which test and jump instructions do not occur. In this section, we introduce straight-line threads with split actions and straight-line threads with split actions and thread forking. These two classes of threads correspond to the two classes of programs: a straight-line thread with split actions is the behaviour of a program of the former class and a straight-line thread with split actions and thread forking is the behaviour of a program of the latter class. For completeness, we introduce straight-line threads as well.

The set $\mathcal{SLT}$ of *straight-line threads* is the subset of $\mathcal{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$ inductively defined as follows:

- if $a \in \mathcal{A}$, then $a \circ \mathsf{D} \in \mathcal{SLT}$ and $a \circ \mathsf{S} \in \mathcal{SLT}$;
- if $a \in \mathcal{A}$ and $p \in \mathcal{SLT}$, then $a \circ p \in \mathcal{SLT}$.

The set $\mathcal{SLT}_{\mathrm{s}}$ of *straight-line threads with split actions* is the subset of $\mathcal{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$ inductively defined as follows:

- if $a \in \mathcal{A}$, then $a \circ \mathsf{D} \in \mathcal{SLT}_{\mathrm{s}}$ and $a \circ \mathsf{S} \in \mathcal{SLT}_{\mathrm{s}}$;
- if $a, b \in \mathcal{A}$, then $a/b \circ \mathsf{D} \in \mathcal{SLT}_{\mathrm{s}}$ and $a/b \circ \mathsf{S} \in \mathcal{SLT}_{\mathrm{s}}$;
- if $a \in \mathcal{A}$ and $p \in \mathcal{SLT}_{\mathrm{s}}$, then $a \circ p \in \mathcal{SLT}_{\mathrm{s}}$;
- if $a, b \in \mathcal{A}$ and $p \in \mathcal{SLT}_{\mathrm{s}}$, then $a/b \circ p \in \mathcal{SLT}_{\mathrm{s}}$.

The set $\mathcal{SLT}_{\mathrm{sf}}$ of *straight-line threads with split actions and thread forking* is the subset of $\mathcal{T}_{\mathrm{TA}^*_{\mathrm{sc}}}$ inductively defined as follows:

- if $a \in \mathcal{A}$, then $a \circ \mathsf{D} \in \mathcal{SLT}_{\mathrm{sf}}$ and $a \circ \mathsf{S} \in \mathcal{SLT}_{\mathrm{sf}}$;
- if $a, b \in \mathcal{A}$, then $a/b \circ \mathsf{D} \in \mathcal{SLT}_{\mathrm{sf}}$ and $a/b \circ \mathsf{S} \in \mathcal{SLT}_{\mathrm{sf}}$;
- if $p \in \mathcal{SLT}_{\mathrm{sf}}$, then $\mathsf{nt}(p) \circ \mathsf{D} \in \mathcal{SLT}_{\mathrm{sf}}$ and $\mathsf{nt}(p) \circ \mathsf{S} \in \mathcal{SLT}_{\mathrm{sf}}$;
- if $a \in \mathcal{A}$ and $p \in \mathcal{SLT}_{\mathrm{sf}}$, then $a \circ p \in \mathcal{SLT}_{\mathrm{sf}}$;
- if $a, b \in \mathcal{A}$ and $p \in \mathcal{SLT}_{\mathrm{sf}}$, then $a/b \circ p \in \mathcal{SLT}_{\mathrm{sf}}$;
- if $p, q \in \mathcal{SLT}_{\mathrm{sf}}$, then $\mathsf{nt}(p) \circ q \in \mathcal{SLT}_{\mathrm{sf}}$.

We have the following inclusions: $\mathcal{SLT} \subset \mathcal{SLT}_{\mathrm{s}} \subset \mathcal{SLT}_{\mathrm{sf}}$, $\mathcal{SLT}_{\mathrm{s}} \subset \mathcal{C}^0$ and $\mathcal{SLT}_{\mathrm{sf}} \subset \mathcal{C}$. Straight-line threads can be described using D, S and action prefixing with atomic actions. For straight-line threads with split actions, split-action prefixing may be used in addition to action prefixing. Split action prefixing is needed to handle program instructions whose processing takes more than one step. For straight-line threads with split actions and thread forking, forking prefixing may be used in addition to action prefixing and split-action prefixing. Forking prefixing is needed to deal with programs that result from parallelization of straight-line programs by use of program forking.

6.2 Algebra of Straight-Line Program with Split Instructions and Forking

We introduce $\mathrm{PGA}_{\mathrm{sl,sf}}$ (ProGram Algebra for Straight-Line programs with Split instructions and Forking). $\mathrm{PGA}_{\mathrm{sl,sf}}$ is a variant of PGA, an algebra of sequential programs based on the idea that sequential programs are in essence sequences of instructions. PGA provides a program notation for threads. A hierarchy of program notations that provide increasingly sophisticated programming features are rooted in PGA (see [6]).

In $\mathrm{PGA}_{\mathrm{sl,sf}}$, it is assumed that there is a fixed but arbitrary set $\mathfrak{A}$ of *basic instructions*. The following *primitive instructions* are taken as constants in $\mathrm{PGA}_{\mathrm{sl,sf}}$:

- for each $a \in \mathfrak{A}$, a *void basic instruction* a;
- for each $a, b \in \mathfrak{A}$, a *split basic instruction* a/b;
- for each closed term P over the signature of $\mathrm{PGA}_{\mathrm{sl,sf}}$, a *fork instruction* $\mathsf{fork}(P)$;
- a *termination instruction* $!$.

We write $\mathfrak{I}$ for the set of all primitive instructions.

In $\mathrm{PGA}_{\mathrm{sl,sf}}$, the test and jump instructions of PGA are absent. This means that, after a primitive instruction of a program other than the termination instruction has been executed, execution of the program always proceeds with the next instruction. After a fork instruction has been executed, in addition, the parallel execution of another program starts up.

The intuition is that the execution of a basic instruction a may modify a state and produces T or F at its completion. In the case of a split basic instruction a/b, a is executed once and next b repeatedly until T is produced. If the execution of a produces T, then b is not at all executed. In the case of a void basic instruction a, simply a is executed once and the value produced is disregarded. The execution of a fork instruction $\mathsf{fork}(P)$ leads to the start-up of the parallel execution of P, and produces the reply T. Execution of the current program proceeds with the next instruction, just like any primitive instruction other than the termination instruction, but it may be affected by the parallel execution of P. The effect of the termination instruction $!$ is that execution terminates.

Qua behaviour, the execution of different programs in parallel that arises from the execution of fork instructions corresponds to synchronous cooperation. This is made precise below by means of a thread extraction operator. The choice for synchronous cooperation is dictated by the intended use of $\mathrm{PGA}_{\mathrm{sl,sf}}$ for investigating program parallelization. In a different context, some kind of interleaving may be chosen instead.

The thread extraction operator defined below, together with the apply operator defined in Section 4.2 make it possible to associate operations of a Maurer machine with basic instructions of $\mathrm{PGA}_{\mathrm{sl,sf}}$.

$\mathrm{PGA}_{\mathrm{sl,sf}}$ has the following constants and operators:

- for each $u \in \mathfrak{I}$, an *instruction* constant u;
- the binary *concatenation* operator $\_\,;\,\_$.

Closed terms over the signature of $\mathrm{PGA}_{\mathrm{sl,sf}}$ are considered to denote finite programs without test and jump instructions. The intuition is that a finite program is in essence a finite non-empty sequence of primitive instructions. That is, programs are considered to be equal if they represent the same finite sequence of

Table 18 Axiom of $\mathrm{PGA}_{\mathrm{sl,sf}}$

$(X\,;Y)\,;Z = X\,;(Y\,;Z)$	PGA1

Table 19 Defining equations for thread extraction operation

$\lvert a\rvert = a \circ \mathsf{D}$	$\lvert a\,;X\rvert = a \circ \lvert X\rvert$
$\lvert a/b\rvert = a/b \circ \mathsf{D}$	$\lvert a/b\,;X\rvert = a/b \circ \lvert X\rvert$
$\lvert\mathsf{fork}(X)\rvert = \mathsf{nt}(\lvert X\rvert) \circ \mathsf{D}$	$\lvert\mathsf{fork}(X)\,;Y\rvert = \mathsf{nt}(\lvert X\rvert) \circ \lvert Y\rvert$
$\lvert !\rvert = \mathsf{S}$	$\lvert !\,;X\rvert = \mathsf{S}$

Table 20 Alphabet axioms for straight-line programs

$\alpha_{\mathrm{slp}}(a) = \{a\}$	$\alpha_{\mathrm{slp}}(a\,;X) = \{a\} \cup \alpha_{\mathrm{slp}}(X)$
$\alpha_{\mathrm{slp}}(a/b) = \{a/b\}$	$\alpha_{\mathrm{slp}}(a/b\,;X) = \{a/b\} \cup \alpha_{\mathrm{slp}}(X)$
$\alpha_{\mathrm{slp}}(\mathsf{fork}(X)) = \alpha_{\mathrm{slp}}(X)$	$\alpha_{\mathrm{slp}}(\mathsf{fork}(X)\,;Y) = \alpha_{\mathrm{slp}}(X) \cup \alpha_{\mathrm{slp}}(Y)$
$\alpha_{\mathrm{slp}}(!) = \emptyset$	$\alpha_{\mathrm{slp}}(!\,;X) = \emptyset$

primitive instructions. Therefore, the only one axiom of $\mathrm{PGA}_{\mathrm{sl,sf}}$ is the one given in Table 18.

Each closed term over the signature of $\mathrm{PGA}_{\mathrm{sl,sf}}$ is considered to denote a program of which the behaviour can be described in $\mathrm{TA}^*_{\mathrm{sc}}$, taking the set $\mathfrak{A}$ of basic instructions for the set $\mathscr{A}$. We define that behaviour by means of the *thread extraction* operation $\lvert\_\rvert$, which assigns a thread to each program. The thread extraction operation is defined by the equations given in Table 19 (for $a, b \in \mathfrak{A}$).

Let P be a closed term over the signature of $\mathrm{PGA}_{\mathrm{sl,sf}}$. The *behaviour* of P, written $[\![P]\!]$, is defined by $[\![P]\!] = \|^{\mathrm{s}}(\langle\lvert P\rvert\rangle)$.

Henceforth, we write $\mathfrak{A}_{\mathrm{s}}$ for the set $\mathfrak{A} \cup \{a/b \mid a, b \in \mathfrak{A}\}$. When investigating program parallelization, it is useful to know the alphabet of a program, i.e. the set of instructions from $\mathfrak{A}_{\mathrm{s}}$ that occur in the program. For that reason, we introduce the *alphabet* operator α_{slp}. The alphabet axioms for straight-line programs with split instructions and forking are given in Table 20.

When investigating program parallelization, it is convenient to use the following extension of the concurrency relation of a Maurer machine.

Given a Maurer machine $H = (M, B, \mathscr{S}, \mathscr{C}, A, [\![\_]\!], C)$, we extend C to $\mathfrak{A}_{\mathrm{s}}$ as follows ($a, a', b, b' \in \mathfrak{A}$):

$$\begin{aligned} C(a/a', b) &\Leftrightarrow C(a,b) \wedge C(a',b)\,, \\ C(a, b/b') &\Leftrightarrow C(a,b) \wedge C(a,b')\,, \\ C(a/a', b/b') &\Leftrightarrow C(a/a', b) \wedge C(a/a', b')\,. \end{aligned}$$

Henceforth, we write $\mathscr{T}_{\mathrm{PGA}_{\mathrm{sl,sf}}}$ for the set of all closed terms over the signature of $\mathrm{PGA}_{\mathrm{sl,sf}}$.

6.3 Relevant Classes of Programs

In Section 6.1, we have introduced straight-line threads, straight-line threads with split actions, and straight-line threads with split actions and thread forking. Here, we introduce the corresponding classes for programs, viz. straight-line programs, straight-line programs with split instructions, and straight-line programs with split instructions and program forking. The last two classes are considered in our study of program parallelization in Section 7.

The set $\mathscr{SLP}$ of *straight-line programs* is the subset of $\mathscr{T}_{\mathrm{PGA}_{\mathrm{sl,sf}}}$ inductively defined as follows:

- $! \in \mathscr{SLP}$;
- if $a \in \mathfrak{A}$, then $a \in \mathscr{SLP}$;
- if $a \in \mathfrak{A}$ and $P \in \mathscr{SLP}$, then $a\,;P \in \mathscr{SLP}$.

The set $\mathscr{SLP}_{\mathrm{s}}$ of *straight-line programs with split instructions* is the subset of $\mathscr{T}_{\mathrm{PGA}_{\mathrm{sl,sf}}}$ inductively defined as follows:

- $! \in \mathscr{SLP}_{\mathrm{s}}$;
- if $a \in \mathfrak{A}$, then $a \in \mathscr{SLP}_{\mathrm{s}}$;
- if $a,b \in \mathfrak{A}$, then $a/b \in \mathscr{SLP}_{\mathrm{s}}$;
- if $a \in \mathfrak{A}$ and $P \in \mathscr{SLP}_{\mathrm{s}}$, then $a\,;P \in \mathscr{SLP}_{\mathrm{s}}$;
- if $a,b \in \mathfrak{A}$ and $P \in \mathscr{SLP}_{\mathrm{s}}$, then $a/b\,;P \in \mathscr{SLP}_{\mathrm{s}}$.

The set $\mathscr{SLP}_{\mathrm{sf}}$ of *straight-line programs with split instructions and program forking* is the subset of $\mathscr{T}_{\mathrm{PGA}_{\mathrm{sl,sf}}}$ inductively defined as follows:

- $! \in \mathscr{SLP}_{\mathrm{sf}}$;
- if $a \in \mathfrak{A}$, then $a \in \mathscr{SLP}_{\mathrm{sf}}$;
- if $a,b \in \mathfrak{A}$, then $a/b \in \mathscr{SLP}_{\mathrm{sf}}$;
- if $P \in \mathscr{SLP}_{\mathrm{sf}}$, then $\mathsf{fork}(P) \in \mathscr{SLP}_{\mathrm{sf}}$;
- if $a \in \mathfrak{A}$ and $P \in \mathscr{SLP}_{\mathrm{sf}}$, then $a\,;P \in \mathscr{SLP}_{\mathrm{sf}}$;
- if $a,b \in \mathfrak{A}$ and $P \in \mathscr{SLP}_{\mathrm{sf}}$, then $a/b\,;P \in \mathscr{SLP}_{\mathrm{sf}}$;
- if $P,Q \in \mathscr{SLP}_{\mathrm{sf}}$, then $\mathsf{fork}(P)\,;Q \in \mathscr{SLP}_{\mathrm{sf}}$.

We have the following inclusions: $\mathscr{SLP} \subset \mathscr{SLP}_{\mathrm{s}} \subset \mathscr{SLP}_{\mathrm{sf}}$. The connection between $\mathscr{SLP}$, $\mathscr{SLP}_{\mathrm{s}}$, $\mathscr{SLP}_{\mathrm{sf}}$ and $\mathscr{SLT}$, $\mathscr{SLT}_{\mathrm{s}}$, $\mathscr{SLT}_{\mathrm{sf}}$ is as follows:

- if $P \in \mathscr{SLP}$ then $|P| \in \mathscr{SLT}$, if $P \in \mathscr{SLP}_{\mathrm{s}}$ then $|P| \in \mathscr{SLT}_{\mathrm{s}}$, if $P \in \mathscr{SLP}_{\mathrm{sf}}$ then $|P| \in \mathscr{SLT}_{\mathrm{sf}}$;
- if $p \in \mathscr{SLT}$ then $p = |P|$ for some $P \in \mathscr{SLP}$, if $p \in \mathscr{SLT}_{\mathrm{s}}$ then $p = |P|$ for some $P \in \mathscr{SLP}_{\mathrm{s}}$, if $p \in \mathscr{SLT}_{\mathrm{sf}}$ then $p = |P|$ for some $P \in \mathscr{SLP}_{\mathrm{sf}}$.

$\mathscr{SLP}_{\mathrm{sf}}$ consists of all P and $P\,;!$ from $\mathscr{T}_{\mathrm{PGA}_{\mathrm{sl,sf}}}$ where $!$ does not occur in P. For all $P \in \mathscr{T}_{\mathrm{PGA}_{\mathrm{sl,sf}}}$, there exists a $P' \in \mathscr{SLP}_{\mathrm{sf}}$ such that $|P| = |P'|$.

Example 6 Suppose that the basic instructions include `LOAD:R1:A`, `LOAD:R2:B`, `ADD:R2:R2:R1` and `STORE:R3:C`. Then the following is a straight-line program:

```
LOAD:R1:A ; LOAD:R2:B ; ADD:R2:R2:R1 ; STORE:R2:C ; !
```

Take the view is that this straight-line program is intended for calculating the sum of the contents of two memory elements and leaving the result of the calculation

behind in a third memory element. That is, suppose that the above-mentioned basic instructions correspond to atomic actions of which the processing amounts to loading the contents of memory element A in register R1, loading the contents of memory element B in register R2, adding the contents of register R1 to the contents of register R2, and storing the contents of register R2 in memory element C. An adaptation of the straight-line program given above, to model that the processing of load instructions takes more than one step, could be the following straight-line program with split instructions:

$$\texttt{LOADI:R1:A/LOADC:R1:A}\,;\texttt{LOADI:R2:B/LOADC:R2:B}\,;\\ \texttt{ADD:R2:R2:R1}\,;\texttt{STORE:R2:C}\,;!$$

A parallellization of this straight-line program with split instructions could be the following straight-line program with split instructions and program forking:

$$\mathsf{fork}(\texttt{LOADI:R2:B/LOADC:R2:B}\,;\texttt{ADD:R2:R2:R1}\,;\texttt{STORE:R2:C}\,;!)\,;\\ \texttt{LOADI:R1:A/LOADC:R1:A}\,;!$$

Getting ahead of our study of program parallelization in Section 7, we mention that this parallelization is not correct if the processing of the split instructions LOADI:R1:A/LOADC:R1:A and LOADI:R2:B/LOADC:R2:B may take different numbers of steps.

In our study of program parallelization, we make the drastic simplification to consider only the parallelization of straight-line programs with split instructions. The reason for that is simply that program parallelization is a complicated matter, which makes it practically necessary to start its study with a drastic simplification. As a case in point, we mention that jump instructions would complicate proving a theorem like Theorem 6, our main theorem about program parallelization, very much.

7 Program Parallelization

In this section, we investigate program parallelization. Our investigation is focused on finding sufficient conditions for the correctness of program parallelizations. We start with presenting some state transformer properties of programs.

7.1 State Transformer Properties of Programs

We present some state transformer properties of straight-line programs with split instructions and program forking which can be useful when investigating program parallelization.

Henceforth, we write $\mathscr{SLP}_{\mathrm{s}}^{\mathrm{wt}}$ for the set $\{P \in \mathscr{SLP}_{\mathrm{s}} \mid \exists P' \in \mathscr{SLP}_{\mathrm{s}} \bullet P' = P\,;!\}$ and $\mathscr{SLP}_{\mathrm{sf}}^{\mathrm{wt}}$ for the set $\{P \in \mathscr{SLP}_{\mathrm{sf}} \mid \exists P' \in \mathscr{SLP}_{\mathrm{sf}} \bullet P' = P\,;!\}$. The superscript wt stands for "without termination".

First, we present a lemma used without mention below in the proofs of Propositions 14, 15 and 16.

Lemma 9

1. *for all* $P \in \mathscr{SLP}_{\mathrm{s}}$, $[\![P]\!] = |P|$;
2. *for all* $P \in \mathscr{SLP}_{\mathrm{s}}$, *there exists a* $p \in \mathscr{C}^0$ *such that* $|P| = p$;
3. *for all* $P \in \mathscr{SLP}_{\mathrm{s}}^{\mathrm{wt}}$ *and* $P' \in \mathscr{SLP}_{\mathrm{sf}}$, $[\![P\,;P']\!] = [\![P\,;!]\!] \cdot [\![P']\!]$.

Proof The first two properties are easily proved by induction on the structure of P. The third property is easily proved by induction on the structure of P, using the first two properties and Proposition 3. □

The following proposition states that state transformer equivalence of the behaviour of programs from $\mathscr{SLP}_{\mathrm{sf}}$ is preserved by prefixing with any program from $\mathscr{SLP}_{\mathrm{s}}$.

Proposition 14 *Let* $P_1 \in \mathscr{SLP}_{\mathrm{s}}$ *and* $P_2, P_2' \in \mathscr{SLP}_{\mathrm{sf}}$. *Then* $[\![P_2]\!] \approx [\![P_2']\!]$ *implies* $[\![P_1\,;P_2]\!] \approx [\![P_1\,;P_2']\!]$.

Proof This follows immediately from Proposition 10. □

The following proposition states that, in every terminating program from $\mathscr{SLP}_{\mathrm{s}}$, a new place can be given to a suffix if each instruction occurring in the suffix can be executed concurrently with each of the instructions occurring between the old place and the new place.

Proposition 15 *Let* $P_1, P_2, P_3 \in \mathscr{SLP}_{\mathrm{s}}^{\mathrm{wt}}$ *be such that* $C(u_2, u_3)$ *for all* $u_2 \in \alpha_{\mathrm{slp}}(P_2)$ *and* $u_3 \in \alpha_{\mathrm{slp}}(P_3)$. *Then* $[\![P_2\,;P_3\,;!]\!] \approx [\![P_3\,;P_2\,;!]\!]$ *and also* $[\![P_1\,;P_2\,;P_3\,;!]\!] \approx [\![P_1\,;P_3\,;P_2\,;!]\!]$.

Proof This follows immediately from Propositions 10 and 11. □

The following proposition states that, in every terminating program from $\mathscr{SLP}_{\mathrm{s}}$, the place of a suffix can be taken by a fork instruction for the suffix that is placed before preceding instructions if those instructions can be executed concurrently with each of the instructions occurring in the suffix.

Proposition 16 *Let* $P_1, P_2, P_3 \in \mathscr{SLP}_{\mathrm{s}}^{\mathrm{wt}}$ *be such that* $C(u_2, u_3)$ *for all* $u_2 \in \alpha_{\mathrm{slp}}(P_2)$ *and* $u_3 \in \alpha_{\mathrm{slp}}(P_3)$. *Then* $[\![P_2\,;P_3\,;!]\!] \approx [\![\mathsf{fork}(P_3\,;!)\,;P_2\,;!]\!]$ *and also* $[\![P_1\,;P_2\,;P_3\,;!]\!] \approx [\![P_1\,;\mathsf{fork}(P_3\,;!)\,;P_2\,;!]\!]$.

Proof By the axioms for synchronous cooperation, we have $[\![P_2\,;P_3\,;!]\!] \approx [\![\mathsf{fork}(P_3\,;!)\,;P_2\,;!]\!]$ iff $[\![P_2\,;P_3\,;!]\!] \approx \|^{\mathrm{s}}(\langle [\![P_2\,;!]\!] \rangle \frown \langle [\![P_3\,;!]\!] \rangle)$. The latter follows immediately from Proposition 12. From this result and Proposition 10, $[\![P_1\,;P_2\,;P_3\,;!]\!] \approx [\![P_1\,;\mathsf{fork}(P_3\,;!)\,;P_2\,;!]\!]$ follows immediately. □

The following proposition states that, for every terminating program from $\mathscr{SLP}_{\mathrm{sf}}$ in which a fork instruction occurs, that fork instruction can be replaced by one for a state transformer equivalent forked program if the instructions occurring in both forked programs can be executed concurrently with each of the instructions occurring after the fork instruction.

Proposition 17 *Let* $P_1, P_3 \in \mathscr{SLP}_{\mathrm{s}}^{\mathrm{wt}}$ *and* $P_2, P_2' \in \mathscr{SLP}_{\mathrm{sf}}^{\mathrm{wt}}$ *be such that* $C(u_2, u_3)$ *for all* $u_2 \in \alpha_{\mathrm{slp}}(P_2) \cup \alpha_{\mathrm{slp}}(P_2')$ *and* $u_3 \in \alpha_{\mathrm{slp}}(P_3)$. *Then* $[\![P_2\,;!]\!] \approx [\![P_2'\,;!]\!]$ *implies* $[\![\mathsf{fork}(P_2\,;!)\,;P_3\,;!]\!] \approx [\![\mathsf{fork}(P_2'\,;!)\,;P_3\,;!]\!]$ *and also* $[\![P_1\,;\mathsf{fork}(P_2\,;!)\,;P_3\,;!]\!] \approx [\![P_1\,;\mathsf{fork}(P_2'\,;!)\,;P_3\,;!]\!]$.

Proof By the axioms for synchronous cooperation, we have $[\![\mathsf{fork}(P_2\,;!)\,;P_3\,;!]\!] \approx [\![\mathsf{fork}(P_2'\,;!)\,;P_3\,;!]\!]$ iff $\|^{\mathrm{s}}(\langle [\![P_3\,;!]\!]\rangle \frown \langle |P_2\,;!|\rangle) \approx \|^{\mathrm{s}}(\langle [\![P_3\,;!]\!]\rangle \frown \langle |P_2'\,;!|\rangle)$. The latter follows immediately from Proposition 13. From this result and Proposition 10, $[\![P_1\,;\mathsf{fork}(P_2\,;!)\,;P_3\,;!]\!] \approx [\![P_1\,;\mathsf{fork}(P_2'\,;!)\,;P_3\,;!]\!]$ follows immediately. □

7.2 Program Partitioning, Annotation and Parallelization

Program parallelization is studied here in the setting of $\mathrm{TA}_{\mathrm{sc}}^{*\bullet}$ and consequently in the scope of the assumption from Section 4.2 that a fixed but arbitrary Maurer machine $H = (M, B, \mathscr{S}, \mathscr{O}, A, [\![\_]\!], C)$ has been given that satisfies certain conditions.

We also use $\mathrm{PGA}_{\mathrm{sl,sf}}$, which offers a convenient program notation for studying program parallelization: the programs of $\mathrm{PGA}_{\mathrm{sl,sf}}$ permit a very direct analysis of semantic issues involved.

We introduce the notion of a partition of a straight-line program, the notion of an annotated partition of a straight-line program, and the notion of the parallelization of a straight-line program induced by an annotated partition of the straight-line program. A straight-line program is a member of $\mathscr{SLP}_{\mathrm{s}}$, whereas a parallelization of a straight-line program is a member of $\mathscr{SLP}_{\mathrm{sf}} \setminus \mathscr{SLP}_{\mathrm{s}}$. Moreover, we introduce a notion of correctness for parallelizations of straight-line programs. The behaviour of a straight-line program and the behaviour of a correct parallelization of that straight-line program are threads that are the same as state transformers.

Let $P \in \mathscr{SLP}_{\mathrm{s}}$ and $P_1, \ldots, P_m \in \mathscr{SLP}_{\mathrm{s}}^{\mathrm{wt}}$. Then $(P_1, \ldots, P_m)$ is a *partition* of P if $P = P_1\,;\ldots;P_m\,;!$.

Let $P = u_1\,;\ldots;u_n$ with $u_1, \ldots, u_n \in \mathfrak{A}_{\mathrm{s}}$ and let $(P_1, \ldots, P_m)$ be a partition of P. Moreover, let $n_0, \ldots, n_m \in \mathbb{N}$ be such that $P_1 = u_{n_0+1}\,;\ldots;u_{n_1}$, $P_2 = u_{n_1+1}\,;\ldots;u_{n_2}$, …, $P_m = u_{n_{m-1}+1}\,;\ldots;u_{n_m}$. Let $l_1, \ldots, l_{m-1} \in \mathbb{N}$. Then $((P_1, \ldots, P_m), (l_1, \ldots, l_{m-1}))$ is an *annotated partition* of P if $n_0 \leq l_1 < n_1$, …, $n_{m-2} \leq l_{m-1} < n_{m-1}$.

Let $P = u_1\,;\ldots;u_n$ with $u_1, \ldots, u_n \in \mathfrak{A}_{\mathrm{s}}$, let $((P_1, \ldots, P_m), (l_1, \ldots, l_{m-1}))$ be an annotated partition of P, and let $n_0, \ldots, n_m \in \mathbb{N}$ be as in the definition of annotated partition above. Let $P_m' = P_m\,;!$ and, for each $i \in [1, m-1]$, let $P_i' = u_{n_{i-1}+1}\,;\ldots;u_{l_i}\,;\mathsf{fork}(P_{i+1}')\,;u_{l_i+1}\,;\ldots;u_{n_i}\,;!$ if $n_{i-1} < l_i$ and $P_i' = \mathsf{fork}(P_{i+1}')\,;u_{l_i+1}\,;\ldots;u_{n_i}\,;!$ if $n_{i-1} = l_i$. Then P_1' is the *parallelization* of P induced by the annotated partition $((P_1, \ldots, P_m), (l_1, \ldots, l_{m-1}))$.

Let $P \in \mathscr{SLP}_{\mathrm{s}}$ and $P' \in \mathscr{SLP}_{\mathrm{sf}}$ be such that P' is the parallelization of P induced by some annotated partition. Then P' is a *correct* parallelization of P if $[\![P]\!] \approx [\![P']\!]$.

If P' is a correct parallelization of P, then $[\![P]\!]$ and $[\![P']\!]$ are the same as state transformers. Moreover, the state transformations that $[\![P]\!]$ can accomplish, $[\![P']\!]$ can accomplish in less steps. That is, $\|([\![P']\!], S)\| < \|([\![P]\!], S)\|$ for all $S \in \mathscr{S}$. Notice that a reduction in number of steps is not guaranteed if we replace $n_0 \leq l_1 < n_1$, …, $n_{m-2} \leq l_{m-1} < n_{m-1}$ by $n_0 \leq l_1 \leq n_1$, …, $n_{m-2} \leq l_{m-1} \leq n_{m-1}$ in the definition of annotated partition.

Program parallelization concerns roughly the following:

– the partitions of a program with at least one annotated version that induces a parallelization of which it can be determined statically that it is a correct one;

– for each such partition, an annotated version that induces a parallelization of which it can be determined statically that it gives the largest reduction in number of steps.

The primary means to determine the above-mentioned correctness and speed-up properties statically is the concurrency relation C.

A sufficient condition for correctness of a parallelization in terms of the concurrency relation C can easily be given.

Theorem 6 *Let $m \geq 2$, and let P, $P_1, \ldots, P_m$, $n_0, \ldots, n_m$, $l_1, \ldots, l_{m-1}$ and $P'_1, \ldots, P'_m$ be as in the definition of parallelization above. Then P'_1 is a correct parallelization of P if, for all $i \in [1, m-1]$, l_i is such that for all $j \in [l_i+1, n_i]$ and $k \in [n_i+1, n_m]$ we have $C(u_j, u_k)$.*

Proof This is easily proved by induction on m, using Propositions 16 and 17. □

7.3 Weaker Sufficient Conditions for Correctness of Parallelizations

Unfortunately, the sufficient condition for correctness of parallelizations given in Theorem 6 is too strong to be useful. However, it can be weakened if there are bounds on the number of steps that the processing of split basic instructions takes. The weakened sufficient condition given in Claim 1 below is not too strong to be useful provided that the diversity of the greatest number of steps that the processing of different instructions take is small.

In Claim 1 below, we use the following notation. Let P, $P_1, \ldots, P_m$, $n_0, \ldots, n_m$, $l_1, \ldots, l_{m-1}$ and $P'_1, \ldots, P'_m$ be as in the definition of parallelization above. Moreover, for each $u \in \mathfrak{I}$, let $ls(u)$ and $gs(u)$ be the least and greatest number of steps that the processing of u takes (if u is not a split basic instruction, then $ls(u) = 1$ and $gs(u) = 1$). Then, for each $i \in [1, m-1]$, we write n'_i for the least n' such that $\sum_{j \in [l_i+1, n_i]} gs(u_j) \leq \sum_{k \in [n_i+1, n']} ls(u_k) - \sum_{j' \in [1, m'_i]} \sum_{k' \in [l_{i+j'}+1, n_{i+j'}]} ls(u_{k'})$, where m'_i is the greatest $m' \in [0, (m-i)-1]$ such that $\sum_{j \in [0, m']} (l_{i+j} - n_{i+j-1}) \leq n_i - n_{i-1}$.[6]

After the presentation of the claim, it will be explained that n'_i is a conservative approximation of the position of the last instruction of P that is possibly executed in concurrency with an instruction of P'_i after P'_{i+1} is forked off, and also that m'_i is one less than the number of programs forked off while P'_i is executed.

Claim 1 *Let $m \geq 2$, and let P, $P_1, \ldots, P_m$, $n_0, \ldots, n_m$, $l_1, \ldots, l_{m-1}$ and $P'_1, \ldots, P'_m$ be as in the definition of parallelization above. Then P'_1 is a correct parallelization of P if, for all $i \in [1, m-1]$, l_i is such that for all $j \in [l_i+1, n_i]$ and $k \in [n_i+1, n'_i]$ we have $C(u_j, u_k)$.*

It can be seen as follows that m'_i (for $i \in [1, m-1]$) is one less than the number of programs forked off while P'_i is executed: if $i+1 \leq m$ then P'_{i+1} is forked off after $l_i - n_{i-1}$ instructions, if $i+2 \leq m$ then P'_{i+2} is forked off after $(l_i - n_{i-1}) + (l_{i+1} - n_i)$ instructions, and so on. In other words, if $i + m' + 1 \leq m$ then $P'_{i+m'+1}$ is forked off after $\sum_{j \in [0, m']} (l_{i+j} - n_{i+j-1})$ instructions. This means that, for $m' \in [0, (m-i)-1]$, if $\sum_{j \in [0, m']} (l_{i+j} - n_{i+j-1}) \leq n_i - n_{i-1}$ then $P'_{i+m'+1}$ is forked off while P'_i is executed.

[6] We use the conventions that $[k, l]$ stands for $\emptyset$ if $k > l$ and $\sum_{i \in I} k_i$ stands for 0 if $I = \emptyset$.

It can be seen as follows that n'_i (for $i \in [1,m-1]$) is a conservative approximation of the position of the last instruction of P that is possibly executed in concurrency with an instruction of P'_i after P'_{i+1} is forked off: $\sum_{k\in[n_i+1,n']} ls(u_k)$ is the least number of steps that it takes to process the instructions of P from the first instruction of P_{i+1} up to and including the instruction with position n' sequentially; and subtraction of $\sum_{j'\in[1,m'_i]} \sum_{k'\in[l_{i+j'}+1,n_{i+j'}]} ls(u_{k'})$ compensates for the instructions of $P'_{i+1}, \ldots, P'_{i+m'_i}$ that are possibly executed in concurrency with instructions of P'_{i+2}, $\ldots$, $P'_{i+m'_i+1}$, namely the instructions of P'_{i+1} executed after P'_{i+2} is forked off and $\ldots$ and the instructions of $P'_{i+m'_i}$ that are executed after $P'_{i+m'_i+1}$ is forked off. This means that if $\sum_{j\in[l_i+1,n_i]} gs(u_j) \leq \sum_{k\in[n_i+1,n']} ls(u_k) - \sum_{j'\in[1,m'_i]} \sum_{k'\in[l_{i+j'}+1,n_{i+j'}]} ls(u_{k'})$ then n' is greater than the position of the last instruction of P that is possibly executed in concurrency with an instruction of P'_i after P'_{i+1} is forked off.

We believe that we can give a proof of Claim 1, but we refrain from giving a proof. Such a proof would involve complicated variants of many of the preceding propositions to be proved. The variants concerned would be attuned to the assumption that for each primitive instruction the least and greatest number of steps that its processing takes are given. We do not consider it realistic to give such a proof in the light of the fact that the weakened sufficient condition is still too strong to be useful if the diversity of the greatest number of steps that the processing of different instructions take is great. This means that the weakened sufficient condition is still rather uninteresting in practice: parallelization is found in techniques for speeding up instruction processing intended to deal with the presence of this diversity.

Given a partition of a straight-line program, we can determine statically which annotated versions of the partition that induce a parallelization satisfying the sufficient condition from Claim 1 give the largest reduction in number of steps. Let P, $P_1,\ldots,P_m$, $n_0,\ldots,n_m$, $l_1,\ldots,l_{m-1}$, $P'_1,\ldots,P'_m$ and $n'_1,\ldots,n'_{m-1}$ be as in Claim 1. If, for all $i \in [1,m-1]$, l_i is such that for all $j \in [l_i+1,n_i]$ and $k \in [n_i+1,n'_i]$ we have $C(u_j,u_k)$ and in addition for all $l' \in [n_{i-1},l_i-1]$ there exist a $j \in [l'+1,l_i]$ and a $k \in [n_i+1,n'_i]$ such that not $C(u_j,u_k)$, then P'_1 is a correct parallelization of P such that for all correct parallelizations P' of P induced by annotated versions of the partition $(P_1,\ldots,P_m)$ that also satisfy the sufficient condition from Claim 1 we have $\|([\![P'_1]\!],S)\| \leq \|([\![P']\!],S)\|$ for all $S \in \mathscr{S}$.

Example 7 Consider the program `P` $=$ `P1;P2;!`, where `P1` and `P2` are as follows:

```
P1 = LOAD:R1:A;        P2 = MOVE:R3:1;
     MUL:R2:R1:R1           MOVE:R4:2;
                            LOAD:R5:B;
                            ADD:R5:R5:R3;
                            MUL:R6:R5:R5;
                            MUL:R6:R6:R4;
                            ADD:R6:R6:R2;
                            STORE:R6:C
```

writing `LOAD:`R`:`M for the split instruction `LOADI:`R`:`M`/LOADC:`R`:`M to increase the resemblance with programs written in some assembly language. `A`, `B` and `C` are different memory elements. If the contents of `A` and `B` are a and b, respectively,

then P calculates $a^2 + 2(b+1)^2$ and stores the result of the calculation in C. It is clear that $(\mathtt{P1}, \mathtt{P2})$ is a partition of P. We suppose that each instruction of P1 may be executed in concurrency with each instruction of P2 except the last but one. The execution of all instructions takes one step, with the exception of the instructions of the form $\mathtt{LOAD}{:}R{:}M$. We suppose that the execution of the latter instructions takes between l and h steps. The annotated partition $((\mathtt{P1}, \mathtt{P2}), (0))$ induces the parallelization $\mathtt{P}' = \mathsf{fork}(\mathtt{P2}\,;\,!)\,;\mathtt{P1}\,;\,!$. This parallelization satisfies the sufficient condition for correctness from Claim 1 provided that $h - l \leq 4$. It is trivial to determine that $((\mathtt{P1}, \mathtt{P2}), (0))$ is the annotated version of $(\mathtt{P1}, \mathtt{P2})$ that gives the largest reduction in number of steps.

8 Conclusions

We have developed an algebraic theory of threads, synchronous cooperation of threads, and interaction of threads with Maurer machines. Setting up a framework in which issues concerning techniques for speeding up instruction processing that involve parallel processing of instructions with diverse variable processing times can be investigated is one of the aims with which we have developed this theory. As part of its development, we have constructed a projective limit model for the theory. In addition to properties of the theory and its projective limit model that are general in nature, we have established properties that are primarily relevant when investigating the issues referred to above.

We have investigated program parallelization, which underlies all explicit multi-threading techniques to speed up instruction processing, using the theory developed. Our finding is that program parallelization, which is done on static grounds, tends to yield marginal speed-ups of instruction processing unless the diversity of greatest processing times is small. The problem is that for all instructions, including the ones with long greatest processing times, the worst case must be taken into account. That leaves little room for provably correct parallelizations that speed up instruction processing substantially.

An obvious idea to reduce the effects of a great diversity of greatest processing times is to use optimistic estimations of processing times for the instructions that take long greatest processing times and to suspend and resume forked-off programs dynamically to compensate for too optimistic estimations of processing times. It is clear that the speed-ups yielded by that highly depend upon the scheduling algorithm used for the resumption of suspended programs and the particular estimations of processing times used. Even if an ideal scheduler is assumed, i.e. one that maximizes simultaneity in the processing of instructions from all programs involved, it appears that there is no clue to the parallelizations that could speed up instruction processing substantially. In fact, the choice of a partition and the choice of an annotated version thereof look to be arbitrary choices now: correctness of the induced parallelizations is not relevant, because it is enforced dynamically, and whether one induced parallelization gives a larger reduction in number of steps than another cannot be determined statically.

We have found that an induction principle to establish state transformer equivalence of infinite threads is material to proving theorems about sufficient conditions for the correctness of program parallelizations. We have also found that, in spite of the drastic simplification made by considering only programs without test

and jump instructions, proving a theorem about a very simple sufficient condition for the correctness of program parallelizations is very difficult. We have not started proving a claim about a somewhat more involved sufficient condition for the correctness of program parallelizations because proving that claim comes very near the limit of what is feasible.

In the area of micro-processor design, explicit-multi-threading is claimed to be a basic technique for speeding up instruction processing substantially. Our main reason to investigate program parallelization was that the arguments that are given for this claim are not soundly based by the standard of theoretical computer science. We also expected to be able to give in the end heuristics for correct program partitioning that speeds up instruction processing substantially. One of our conclusions from the results of the investigation of program parallelization is that the justness of the claim is far less evident than it is generally assumed in the area of micro-processor design. Another conclusion from the results of our investigation is that the development of useful heuristics is as yet practically unfeasible.

In this paper, we have carried on the line of research that has already resulted in [8–10]. We pursue with this line of research the object to develop an approach to design new micro-architectures that allows for their correctness and anticipated speed-up results to be verified. It emanates from the work presented in [6, 3]. There is another related line of research that emanates from that work. That line of research concerns the development of a theory about threads, multi-threading and interaction of threads with services that is useful for gaining insight into the semantic issues concerning the multi-threading related features found in contemporary programming languages. It has already resulted in [7, 11, 14, 12, 13]. We believe that the theory being developed may also be useful when developing parallelization techniques for compilers that have to take care of program parallelization for programs written in programming languages such as Java and C#.

Acknowledgements We thank two anonymous referees for suggesting improvements of the presentation of the paper.

A CPO Structure for Projective Limit Model

In this appendix, we make A^∞ into a complete partial ordering (cpo) to establish the existence of least solutions of recursion equations using Tarski's fixed point theorem.

The *approximation* relation $\sqsubseteq \subseteq A_\omega \times A_\omega$ is the smallest partial ordering such that for all $p, p', q, q' \in A_\omega$:

- $\mathsf{D} \sqsubseteq p$;
- $p \sqsubseteq p' \Rightarrow \mathsf{tau} \circ p \sqsubseteq \mathsf{tau} \circ p'$;
- for all $\xi \in \mathscr{BA}$, $p \sqsubseteq p' \wedge q \sqsubseteq q' \Rightarrow p \trianglelefteq \xi \trianglerighteq q \sqsubseteq p' \trianglelefteq \xi \trianglerighteq q'$;
- $p \sqsubseteq p' \wedge q \sqsubseteq q' \wedge r \sqsubseteq r' \Rightarrow p \trianglelefteq \mathsf{nt}(r) \trianglerighteq q \sqsubseteq p' \trianglelefteq \mathsf{nt}(r') \trianglerighteq q'$;
- for all $\xi \in \mathscr{BA}$, $p \sqsubseteq p' \wedge q \sqsubseteq q' \Rightarrow p \lhd \mathsf{y}_\xi \rhd q \sqsubseteq p' \lhd \mathsf{y}_\xi \rhd q'$.

The *approximation* relation $\sqsubseteq \subseteq A^\infty \times A^\infty$ is defined component-wise:

$$(p_n)_{n\in\mathbb{N}} \sqsubseteq (q_n)_{n\in\mathbb{N}} \Leftrightarrow \forall n \in \mathbb{N} \bullet p_n \sqsubseteq q_n \,.$$

The approximation relation $\sqsubseteq$ on A_n is simply the restriction of $\sqsubseteq$ on A_ω to A_n.

The following proposition states that any $p \in A_\omega$ is finitely approximated by projection.

Proposition 18 *For all $p \in A_\omega$:*

$$\exists n \in \mathbb{N} \bullet (\forall k < n \bullet \pi_k(p) \sqsubseteq \pi_{k+1}(p) \wedge \forall l \geq n \bullet \pi_l(p) = p) \,.$$

Proof The proof follows the same line as the proof of Proposition 1 from [3]. This means that it is a rather trivial proof by induction on the structure of p. Here, we have to consider the additional cases $p \equiv p' \trianglelefteq \mathsf{nt}(p''') \trianglerighteq p''$ and $p \equiv p' \lhd \mathsf{y}_\xi \rhd p''$. These cases go analogously to the case $p \equiv p' \trianglelefteq \xi \trianglerighteq p''$. □

The properties stated in the following lemma will be used in the proof of Theorem 7 given below.

Lemma 10 *For all $n \in \mathbb{N}$:*

1. *$(A_n, \sqsubseteq)$ is a cpo;*
2. *π_n is continuous;*
3. *for all $p \in A_\omega$:*
   (a) *$\pi_n(p) \sqsubseteq p$;*
   (b) *$\pi_n(\pi_n(p)) = \pi_n(p)$;*
   (c) *$\pi_{n+1}(\pi_n(p)) = \pi_n(p)$.*

Proof The proof follows similar lines as the proof of Proposition 2 from [3]. Property 1 follows from the fact that every directed set $P \subseteq A_n$ is finite. Like in [3], this fact is proved by induction on n. Due to the presence of reply conditionals, the proof is more involved. It is the only proof in this paper that makes use of the assumption that $\mathscr{AA}$ is a finite set. For Property 2, we now have to use induction on the structure of the elements of A_ω and distinction between the cases $n = 0$ and $n > 0$ for non-forking and forking postconditional compositions. Due to the presence of reply conditionals, we cannot use induction on n and case distinction on the structure of the elements of A_ω like in [3]. However, the crucial details of the proof remain the same. Like in [3], Property 3a follows immediately from Proposition 18. Properties 3b and 3c follow immediately from Lemma 4. □

The following theorem states some basic properties of the approximation relation $\sqsubseteq$ on A^∞.

Theorem 7 *$(A^\infty, \sqsubseteq)$ is a cpo with $\bigsqcup P = (\bigsqcup\{\pi_n(p) \mid p \in P\})_{n\in\mathbb{N}}$ for all directed sets $P \subseteq A^\infty$. Moreover, up to (order) isomorphism $A_\omega \subseteq A^\infty$.*

Proof The proof follows the same line as the proof of Theorem 1 from [3]. That is, using general properties of the projective limit construction on cpos, the first part follows immediately from Properties 1 and 2 from Lemma 10, and the second part follows easily from Proposition 18 and Property 3 from Lemma 10. □

Another important property of the approximation relation $\sqsubseteq$ on A^∞ is stated in the following theorem.

Theorem 8 *The operations from the projective limit model for $\mathrm{TA}^*_{\mathrm{sc}}$ are continuous with respect to $\sqsubseteq$.*

Proof With the exception of the conditional action repetition operations, the proof follows the same line for all kinds of operations. It begins by establishing the monotonicity of the operation on A_ω. For the non-forking and forking postconditional composition operations and the reply conditional operations, this follows immediately from the definition of $\sqsubseteq$ on A_ω. For the synchronous cooperation operation, it is straightforwardly proved by induction on $\nu(p)$ and case distinction according to the left-hand sides of the axioms for synchronous cooperation. Then the monotonicity of the operations on A^∞ follows from their monotonicity on A_ω, the monotonicity of the projection operations and the definition of $\sqsubseteq$ on A^∞.

For the conditional action repetition operations, the proof differs in that it begins with establishing – with a proof by induction on n, using axioms for conditional action repetition – that, for all $p, q \in A_\omega$, for all $n \in \mathbb{N}$, $p \sqsubseteq q$ implies $\pi_n(\xi^{*b} p) \sqsubseteq \pi_n(\xi^{*b} q)$. From this and the definition of $\sqsubseteq$ on A^∞, the monotonicity of the conditional action repetition operations on A^∞ follows as well.

What remains to be proved is that least upper bounds of directed sets are preserved by the operations. We will show how the proof goes for the non-forking postconditional composition operations. The proofs for the other kinds of operations go similarly. Let $P, Q \subseteq A^\infty$ be directed sets. Then, for all $n \in \mathbb{N}$, $\{\pi_n(p) \mid p \in P\}, \{\pi_n(q) \mid q \in Q\}, \{\pi_n(p) \trianglelefteq \xi \trianglerighteq \pi_n(q) \mid p \in P \wedge q \in Q\} \subseteq A_n$ are directed sets by the monotonicity of π_n. It is easily proved by induction on n, using the definition of $\sqsubseteq$ on A_n, that these directed sets are finite. This implies that they have maximal elements. From this, it follows by the monotonicity of $\_ \trianglelefteq \xi \trianglerighteq \_$ that, for all $n \in \mathbb{N}$, $(\bigsqcup\{\pi_n(p) \mid p \in P\}) \trianglelefteq \xi \trianglerighteq (\bigsqcup\{\pi_n(q) \mid q \in Q\}) = \bigsqcup\{\pi_n(p) \trianglelefteq \xi \trianglerighteq \pi_n(q) \mid p \in P \wedge q \in Q\}$. From this, it follows by the property of lubs of directed sets stated in Theorem 7 and the definition of π_{n+1} that, for all $n \in \mathbb{N}$, $\pi_{n+1}((\bigsqcup P) \trianglelefteq \xi \trianglerighteq (\bigsqcup Q)) = \pi_{n+1}(\bigsqcup\{p \trianglelefteq \xi \trianglerighteq q \mid p \in P \wedge q \in Q\})$. Because $\pi_0((\bigsqcup P) \trianglelefteq \xi \trianglerighteq (\bigsqcup Q)) = \mathsf{D} = \pi_0(\bigsqcup\{p \trianglelefteq \xi \trianglerighteq q \mid p \in P \wedge q \in Q\})$, also for all $n \in \mathbb{N}$, $\pi_n((\bigsqcup P) \trianglelefteq \xi \trianglerighteq (\bigsqcup Q)) = \pi_n(\bigsqcup\{p \trianglelefteq \xi \trianglerighteq q \mid p \in P \wedge q \in Q\})$. From this, it follows by the definition of $\sqsubseteq$ on A^∞ that $(\bigsqcup P) \trianglelefteq \xi \trianglerighteq (\bigsqcup Q) = \bigsqcup\{p \trianglelefteq \xi \trianglerighteq q \mid p \in P \wedge q \in Q\}$. □

We have the following important result about recursion equations.

Theorem 9 *Let $x \in \mathscr{X}$, let $P \subseteq A^\infty$, and let $t \in \mathscr{T}_P^{\{x\}}$. Then the recursion equation $x = t$ has a least solution with respect to $\sqsubseteq$, i.e. there exists a $p \in A^\infty$ such that $[\![t]\!]^x(p) = p$ and, for all $q \in A^\infty$, $[\![t]\!]^x(q) = q$ implies $p \sqsubseteq q$.*

Proof We have from Theorem 7 that $(A^\infty, \sqsubseteq)$ is a cpo and, using Theorem 8, it is easily proved by induction on the structure of t that $[\![t]\!]^x$ is continuous. From this, we conclude by Tarski's fixed point theorem that there exists a $p \in A^\infty$ such that $[\![t]\!]^x(p) = p$ and, for all $q \in A^\infty$, $[\![t]\!]^x(q) = q$ implies $p \sqsubseteq q$. Hence, the recursion equation $x = t$ has a least solution with respect to $\sqsubseteq$. □

The following proposition relates the ordering relation $\sqsubseteq$ introduced in this appendix with the ordering relation $\underset{\sim}{\sqsubseteq}$ introduced in Section 5.1.

Proposition 19 *For all $p, q \in A^\infty$, $p \sqsubseteq q \Rightarrow p \underset{\sim}{\sqsubseteq} q$.*

Proof Let $p, q \in A^\infty$ be such that $p \sqsubseteq q$. Then, for all $n \in \mathbb{N}$, we have $\pi_n(p) \sqsubseteq \pi_n(q)$ by the monotonicity of π_n. It is easily proved by induction on the structure of p' that $p' \sqsubseteq q'$ implies $p' \underset{\sim}{\sqsubseteq} q'$ for all $p', q' \in A_\omega$. Hence, for all $n \in \mathbb{N}$, we have $\pi_n(p) \underset{\sim}{\sqsubseteq} \pi_n(q)$ as well. From this, it follows immediately that, for all $n \in \mathbb{N}$, there exists an $m \in \mathbb{N}$ such that $\pi_n(p) \underset{\sim}{\sqsubseteq} \pi_m(q)$. From this, it follows by the state transformer inclusion principle (see Table 16) that $p \underset{\sim}{\sqsubseteq} q$. □

We have the following corollary concerning $\underset{\sim}{\sqsubseteq}$ from Propositions 18 and 19.

Corollary 3 *For all $p \in A_\omega$:*

$$\exists n \in \mathbb{N} \bullet (\forall k < n \bullet \pi_k(p) \underset{\sim}{\sqsubseteq} \pi_{k+1}(p) \wedge \forall l \geq n \bullet \pi_l(p) \approx p) \,.$$

B Glossary of Symbols

In this appendix, we provide a glossary of symbols used in this paper.

Notation	*Meaning*	*Page*
Thread algebras		
BTA_δ	basic thread algebra with blocking	4
$\mathrm{TA}_{\mathrm{sc}}$	thread algebra with synchronous cooperation	5
$\mathrm{TA}_{\mathrm{sc}}^*$	$\mathrm{TA}_{\mathrm{sc}}$ with conditional action repetition	10
$\mathrm{TA}_{\mathrm{sc}}^{*\bullet}$	$\mathrm{TA}_{\mathrm{sc}}^*$ with thread to Maurer machine application	25

Thread algebra notation

D	deadlock	4
S	termination	4
$p \trianglelefteq \xi \trianglerighteq q$	non-forking postconditional composition	4
$\xi \circ p$	action prefixing	4
$\xi \,\&\, \xi'$	synchronization	5
$\|\|^{\mathsf{s}}(\langle p_1\rangle \frown \ldots \frown \langle p_n\rangle)$	synchronous cooperation	6
$p \trianglelefteq \mathsf{nt}(r) \trianglerighteq q$	forking postconditional composition	6
$p \triangleleft \mathsf{y}_\xi \triangleright q$	reply conditional	6
$\mathsf{nt}(p) \circ q$	forking prefixing	6
$\xi \,{}^{*b}\, p$	conditional action repetition	10
$\xi / \xi' \circ p$	split-action prefixing	10
$p \cdot q$	p with all occurrences of S replaced by q	10
$\pi_n(p)$	projection	11
$\alpha(p)$	alphabet	13
$p \bullet S$	apply	25

Sets of actions

$\mathscr{BA}$	set of basic actions	4
$\mathscr{A}$	set of basic actions and tau	4
$\mathscr{A}_\delta$	set of basic actions, tau and δ	4
$\mathscr{AA}$	set of atomic actions	5
$\mathscr{CA}$	set of concurrent actions	5
$\mathscr{CA}_\delta$	set of concurrent actions and δ	5

Sets of terms

$\mathscr{T}_{\mathrm{TA_{sc}}}$	set of closed terms over signature of $\mathrm{TA_{sc}}$	8
$\mathscr{B}$	set of basic terms	8
$\mathscr{B}^0$	set of basic terms without forking	8
$\mathscr{T}_{\mathrm{TA^*_{sc}}}$	set of closed terms over signature of $\mathrm{TA^*_{sc}}$	10
$\mathscr{C}$	set of semi-basic terms	10
$\mathscr{C}^0$	set of semi-basic terms without forking	10
$\mathscr{SLT}$	set of straight-line threads	34
$\mathscr{SLT}_{\mathrm{s}}$	set of straight-line threads with split actions	34
$\mathscr{SLT}_{\mathrm{sf}}$	set of straight-line threads with split actions and thread forking	34

Domains of models

A_ω	domain of initial model for $\mathrm{TA_{sc}}$	17
A^∞	domain of projective limit model for $\mathrm{TA_{sc}}$	18

Maurer machines

M	memory	24
B	base set	24
$\mathscr{S}$	set of states	24
$\mathscr{O}$	set of operations	24
A	set of atomic actions	25
$[\![\_]\!]$	atomic action interpretation function	25
C	atomic action concurrency relation	25
$\uparrow$	undefined state	27
$\|(p,S)\|$	length of computation	28

State transformer equivalence

$\approx$	state transformer equivalence	29
$\sqsubset\!\!\sim$	state transformer inclusion	30

Program algebra

$\mathrm{PGA}_{\mathrm{sl,sf}}$	straight-line program algebra with split instructions and forking	35
$\mathfrak{A}$	set of basic instructions	35
$\mathfrak{J}$	set of primitive instructions	35
$\mathfrak{A}_{\mathrm{s}}$	set of void and split basic instructions	36
a	void basic instruction	35
a/b	split basic instruction	35
$\mathsf{fork}(P)$	fork instruction	35
!	termination instruction	35
$P\,;Q$	concatenation	35
$\alpha_{\mathrm{slp}}(P)$	alphabet	36
$\lvert P\rvert$	thread extraction	36
$[\![P]\!]$	program behaviour	36
$\mathscr{T}_{\mathrm{PGA}_{\mathrm{sl,sf}}}$	set of closed terms over signature of $\mathrm{PGA}_{\mathrm{sl,sf}}$	36
$\mathscr{SLP}$	set of straight-line programs	37
$\mathscr{SLP}_{\mathrm{s}}$	set of straight-line programs with split instructions	37
$\mathscr{SLP}_{\mathrm{sf}}$	set of straight-line programs with split instructions and forking	37

References

1. de Bakker, J.W., Bergstra, J.A., Klop, J.W., Meyer, J.J.C.: Linear time and branching time semantics for recursion with merge. Theoretical Computer Science **34**, 135–156 (1984)
2. de Bakker, J.W., Zucker, J.I.: Processes and the denotational semantics of concurrency. Information and Control **54**(1/2), 70–120 (1982)
3. Bergstra, J.A., Bethke, I.: Polarized process algebra and program equivalence. In: J.C.M. Baeten, J.K. Lenstra, J. Parrow, G.J. Woeginger (eds.) Proceedings 30th ICALP, *Lecture Notes in Computer Science*, vol. 2719, pp. 1–21. Springer-Verlag (2003)
4. Bergstra, J.A., Klop, J.W.: Process algebra for synchronous communication. Information and Control **60**(1/3), 109–137 (1984)
5. Bergstra, J.A., Loots, M.E.: Program algebra for component code. Formal Aspects of Computing **12**(1), 1–17 (2000)
6. Bergstra, J.A., Loots, M.E.: Program algebra for sequential code. Journal of Logic and Algebraic Programming **51**(2), 125–156 (2002)
7. Bergstra, J.A., Middelburg, C.A.: Thread algebra for strategic interleaving. To appear in *Formal Aspects of Computing*. Preliminary version: Computer Science Report 04-35, Department of Mathematics and Computer Science, Eindhoven University of Technology (2004)
8. Bergstra, J.A., Middelburg, C.A.: Maurer computers with single-thread control. To appear in *Fundamenta Informaticae*. Preliminary version: Computer Science Report 05-17, Department of Mathematics and Computer Science, Eindhoven University of Technology (2005)
9. Bergstra, J.A., Middelburg, C.A.: Simulating Turing machines on Maurer machines. To appear in *Journal of Applied Logic*. Preliminary version: Computer Science Report 05-28, Department of Mathematics and Computer Science, Eindhoven University of Technology (2005)
10. Bergstra, J.A., Middelburg, C.A.: Maurer computers for pipelined instruction processing. To appear in *Mathematical Structures in Computer Science*. Preliminary version: Computer Science Report 06-12, Department of Mathematics and Computer Science, Eindhoven University of Technology (2006)
11. Bergstra, J.A., Middelburg, C.A.: Thread algebra with multi-level strategies. Fundamenta Informaticae **71**(2/3), 153–182 (2006)
12. Bergstra, J.A., Middelburg, C.A.: A thread calculus with molecular dynamics. Computer Science Report 06-24, Department of Mathematics and Computer Science, Eindhoven University of Technology (2006)
13. Bergstra, J.A., Middelburg, C.A.: Distributed Strategic Interleaving with Load Balancing. Computer Science Report 07-03, Department of Mathematics and Computer Science, Eindhoven University of Technology (2007)

14. Bergstra, J.A., Middelburg, C.A.: A thread algebra with multi-level strategic interleaving. Theory of Computing Systems, **41**(1), 3–32 (2007)
15. Bergstra, J.A., Ponse, A.: Combining programs and state machines. Journal of Logic and Algebraic Programming **51**(2), 175–192 (2002)
16. Bolychevsky, A., Jesshope, C.R., Muchnick, V.: Dynamic scheduling in RISC architectures. IEE Proceedings Computers and Digital Techniques **143**(5), 309–317 (1996)
17. Croom, F.H.: Principles of Topology. Saunders College Publishing, Philadelphia (1989)
18. Dugundji, J.: Topology. Allyn and Bacon, Boston (1966)
19. Hodges, W.A.: Model Theory, *Encyclopedia of Mathematics and Its Applications*, vol. 42. Cambridge University Press, Cambridge (1993)
20. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation, second edn. Addison-Wesley, Reading, MA (2001)
21. Jesshope, C.R., Luo, B.: Micro-threading: A new approach to future RISC. In: ACAC 2000, pp. 34–41. IEEE Computer Society Press (2000)
22. Kranakis, E.: Fixed point equations with parameters in the projective model. Information and Computation **75**(3), 264–288 (1987)
23. Maurer, W.D.: A theory of computer instructions. Journal of the ACM **13**(2), 226–235 (1966)
24. Maurer, W.D.: A theory of computer instructions. Science of Computer Programming **60**, 244–273 (2006)
25. Mousavi, M.R., Gabbay, M.J., Reniers, M.A.: SOS for higher order processes. In: M. Abadi, L. de Alfaro (eds.) CONCUR 2005, *Lecture Notes in Computer Science*, vol. 3653, pp. 308–322. Springer-Verlag (2005)
26. Mousavi, M.R., Reniers, M.A., Groote, J.F.: Notions of bisimulation and congruence formats for SOS with data. Information and Computation **200**, 107–147 (2005)
27. Schmidt, D.A.: Denotational Semantics: A Methodology for Language Development. Allyn and Bacon, Boston (1986)
28. Stoltenberg-Hansen, V., Tucker, J.V.: Algebraic and fixed point equations over inverse limits of algebras. Theoretical Computer Science **87**, 1–24 (1991)
29. Ungerer, T., Robič, B., Šilc, J.: A survey of processors with explicit multithreading. ACM Computing Surveys **35**(1), 29–63 (2003)
30. Vu, T.D.: Metric denotational semantics for BPPA. Report PRG0503, Programming Research Group, University of Amsterdam (2005)