Fundamenta Informaticae 00 (0000) 1–30
IOS Press

Maurer Computers with Single-Thread Control[*]

J.A. Bergstra[†] and C.A. Middelburg[‡] [C]
Programming Research Group
University of Amsterdam
P.O. Box 41882, 1009 DB Amsterdam, the Netherlands
J.A.Bergstra@uva.nl, C.A.Middelburg@uva.nl

Abstract. We investigate basic issues concerning stored threads and their execution, building upon Maurer's model for computers and the thread algebra of Bergstra et al. We show among other things that a single thread can control the execution on a Maurer machine of any executable finite-state thread stored in the memory of the Maurer machine. We also relate stored threads with programs as considered in the program algebra of Bergstra et al. The work is intended as a preparation for the development of a formal approach to model micro-architectures and to verify their correctness and anticipated speed-up results.

Keywords: Maurer computer, thread algebra, stored thread, execution, control thread, program algebra

1. Introduction

In this paper, we study the feasibility of an approach based on Maurer machines and Basic Thread Algebra (BTA) to model micro-architectures and to verify their correctness and anticipated speed-up results. In particular, we investigate basic issues concerning stored threads and their execution.

Maurer machines are based on a model for computers proposed by Maurer in [19], a paper from 40 years ago. Maurer's model for computers is quite different from the well-known models such as register

[*]This research was partly carried out in the framework of the GLANCE-project MICROGRIDS, which is funded by the Netherlands Organisation for Scientific Research (NWO).
[†]Second affiliation: Department of Philosophy, Utrecht University, Utrecht, the Netherlands.
[‡]Second affiliation: Division of Computer Science, Eindhoven University of Technology, Eindhoven, the Netherlands.
[C]Corresponding author

machines, multi-stack machines and Turing machines (see e.g. [17]). The strength of Maurer's model is that it is close to real computers. BTA is a form of process algebra which is introduced in [6] under the name Basic Polarized Process Algebra (BPPA). It 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. To make it possible that threads direct a Maurer machine in performing operations on its state, BTA is extended in this paper with, for each Maurer machine, an apply operator. Applying a thread to a Maurer machine amounts to generating a sequence of state changes according to the operations that the Maurer machine associates with the basic actions performed by the thread.

Why did we choose to use Maurer machines and BTA as the basis of an approach to model micro-architectures? First of all, well-known models for computers, such as register machines, multi-stack machines and Turing machines, are too general for our purpose. Unlike Maurer's model for computers, those models have little in common with real computers. They abstract from many aspects of real computers with which the design of a micro-architectures must deal. Maurer's model for computers 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. The design of micro-architectures must deal with these aspects of computers. Secondly, well-known process algebras, such as ACP [2], CCS [22], and CSP [16], are too general for our purpose as well. BTA has been designed as an algebra of deterministic sequential processes that interact with a machine. In [11], we show that the processes considered in BTA can be viewed as processes that are definable over an extension of ACP with conditions introduced in [9]. However, it is quite awkward to describe and analyze processes of this kind using such a general process algebra.

In this paper, we show step by step that a single thread can control the execution on a Maurer machine of any executable finite-state thread stored in the memory of the Maurer machine. By that we give a theoretical underpinning of basic ideas from the practice of micro-architectures. At the same time, we demonstrate, admittedly by means of a very simple micro-architecture, that a micro-architecture can be taken as a provably correct refinement of a more abstract architecture, possibly an instruction set architecture, when using the approach based on Maurer machines and basic thread algebra. In [8], we make use of the experience gained with the work presented in this paper to model a more advanced micro-architecture, namely a micro-architecture with pipelined instruction processing, and to verify its correctness.

Using some kind of strategic interleaving, several single-thread controlled Maurer machines can be put in parallel, which might be relevant to the design of multiprocessor architectures. Several kinds of strategic interleaving have been elaborated in earlier work, see e.g. [13, 7]. Using the simplest kind of strategic interleaving, called cyclic interleaving, we show also in this paper that finite-state threads of arbitrary size can be dealt with if the Maurer machine on which the execution takes place leaves the fetching of the basic actions to another Maurer machine whose memory size is sufficient for the thread concerned.

We also demonstrate in this paper that there is a close connection between stored threads and PGLD programs. PGLD is one of the program notations based on Program Algebra (PGA) introduced in [6]. It is close to existing assembly languages, and the behaviour of PGLD programs are threads of the kind considered in BTA. What is important for the modelling of micro-architectures is the presence of test and jump instructions in PGLD. The modelling of more advanced micro-architectures must more often than not deal explicitly with test and jump instructions (cf. [8]). This makes stored threads often less adequate when modelling more advanced micro-architectures. In such cases, conversion from stored threads to stored PGLD programs is a feasible option.

The work presented in this paper, as well as the work presented in [8], has convinced us that a special notation for the description of micro-architectures is desirable. For example, it is annoying that, for each memory element that is not affected by an operation, this must be described explicitly. However, we found that fixing an appropriate notation still requires some significant design decisions.

As mentioned above, Maurer's model for computers is quite different from Turing's model. The latter model belongs to the foundations of theoretical computer science, whereas the model used in our approach to model micro-architectures is relatively unknown indeed. In order to acquire more insight into the connections between Turing machines, Maurer computers and real computers, we investigate ways to simulate Turing machines on Maurer computers in [12].

The structure of this paper is as follows. First of all, we review Maurer's model for computers (Section 2) and BTA (Section 3). Following this, we introduce operators which allow for threads to direct Maurer machines in performing operations on their state (Section 4). After that, we enhance Maurer machines step by step till we have attained the result that a single thread can control the execution on a Maurer machines of any executable finite-state thread stored in the memory of the Maurer machines (Sections 5–8). We demonstrate that such control can be accomplished with a single control operation as well (Section 9). Next, we introduce parallel composition of Maurer machines and cyclic interleaving of threads (Section 10) and show that finite-state threads of arbitrary size can be dealt with (Section 11). Then, we relate PGLD programs to stored threads (Section 12). Finally, we make some concluding remarks (Section 13).

2. Maurer's Model for Computers

In this section, we shortly review the model for computers proposed by Maurer in [19]. We use the phrase Maurer computer for what is a computer according to Maurer's definition.

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

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

and satisfies the following conditions:

- if $S_1, S_2 \in \mathcal{S}$, $M' \subseteq M$ 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 $S_3 \in \mathcal{S}$;
- if $S_1, S_2 \in \mathcal{S}$, 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 $\mathcal{S}$ are called the *states*, and the members of $\mathcal{O}$ are called the *operations*. It is obvious that the first condition is satisfied if C is *complete*, i.e. if $\mathcal{S}$ 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 [19], operations are called instructions. We use the term operation because of the confusion that would otherwise arise with the more established use of the term instruction in the area of computer systems architecture and organization.

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. Thus, a Maurer computer has much in common with a real computer. The first condition on the states of a Maurer computer is a structural condition and the second one is a finite variability condition. The following theorem from [19] gives an interesting characterization of the set of states of a Maurer computer.

Theorem 2.1. Let $(M, B, \mathcal{S}, \mathcal{O})$ be a Maurer computer, let $S_0 \in \mathcal{S}$, and let $B_x = \{b \in B \mid \exists S \in \mathcal{S} \bullet S(x) = b\}$ for all $x \in M$. Then $\mathcal{S}$ is the set of all functions $S : M \to B$ such that $S(x) \in B_x$ for all $x \in M$ and $\{x \in M \mid S_0(x) \neq S(x)\}$ is finite.

Let $(M, B, \mathcal{S}, \mathcal{O})$ be a Maurer computer, and let $O : \mathcal{S} \to \mathcal{S}$. 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:[1]

$$OR(O) = \big\{x \in M \bigm| \exists S \in \mathcal{S} \bullet S(x) \neq O(S)(x)\big\} ,$$

$$\begin{aligned} IR(O) = \big\{x \in M \bigm| \exists S_1, S_2 \in \mathcal{S} \bullet (\forall z \in M \setminus \{x\} \bullet S_1(z) = S_2(z) \land {} \\ \exists y \in OR(O) \bullet O(S_1)(y) \neq O(S_2)(y))\big\} . \end{aligned}$$

$OR(O)$ 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. The following theorem from [19] gives a fundamental property of the input region and the output region of an operation.

Theorem 2.2. Let $(M, B, \mathcal{S}, \mathcal{O})$ be a Maurer computer, let $S_1, S_2 \in \mathcal{S}$ and $O \in \mathcal{O}$. Then $S_1 \restriction IR(O) = S_2 \restriction IR(O)$ implies $O(S_1) \restriction OR(O) = O(S_2) \restriction OR(O)$.[2]

In words, every operation transforms states that coincide on the input region of the operation to states that coincide on the output region of the operation. The second condition on the states of a Maurer computer is necessary for this property to hold. The first condition on the states of a Maurer computer could be relaxed somewhat (for more details, see [19]).

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

$$\begin{aligned} AR(M'', O) = \big\{x \in OR(O) \bigm| \exists S_1, S_2 \in \mathcal{S} \bullet (\forall z \in IR(O) \setminus M'' \bullet S_1(z) = S_2(z) \land {} \\ O(S_1)(x) \neq O(S_2)(x))\big\} , \end{aligned}$$

$$RA(M', O) = \big\{x \in IR(O) \bigm| AR(\{x\}, O) \cap M' \neq \emptyset\big\} .$$

[1]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 $\land$ and $\lor$, and $\land$ and $\lor$ bind stronger than $\Rightarrow$ and $\Leftrightarrow$. Quantifiers are given the smallest possible scope.

[2]We use the notation $f \quad D$, where f is a function and $D \subseteq dom(f)$, for the function g with $dom(g) = D$ such that for all $d \in dom(g)$, $g(d) = f(d)$.

$AR(M'', O)$ 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 [19], Maurer gives many results about the composition of operations, the decomposition of operations and the existence of operations with specified input, output and affected regions. In Appendix A, we summarize the main results. Recently, a revised and expanded version of [19], which includes all the proofs, has appeared in [20].

3. Basic Thread Algebra

In this section, we review Basic Thread Algebra (BTA), 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, it is assumed that there is a fixed but arbitrary set of *basic actions* $\mathcal{A}$. BTA has the following constants and operators:

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

It is assumed that there are infinitely many variables, including x and y. Terms of BTA are built as usual from these variables and the constants and operators of BTA. We use infix notation for postconditional composition. We introduce *action prefixing* as an abbreviation: $a \circ p$, where p is a term of BTA, 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. On 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 of BTA. 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).

A *recursive specification* over BTA is a set of equations $E = \{X = t_X \mid X \in V\}$, where V is a set of variables and each t_X is a term of BTA that only contains variables from V. We write $\mathrm{V}(E)$ for the set of all variables that occur on the left-hand side of an equation in E. Let t be a term of BTA containing a variable X. Then an occurrence of X in t is *guarded* if t has a subterm of the form $t' \trianglelefteq a \trianglerighteq t''$ containing this occurrence of X. A recursive specification E is *guarded* if all occurrences of variables in the right-hand sides of its equations are guarded or it can be rewritten to such a recursive specification using the equations of E. We are only interested in models of BTA in which guarded recursive specifications have unique solutions, such as the projective limit model of BTA presented in [3, 6]. A thread that is the solution of a finite guarded recursive specification over BTA is called a *finite-state* thread.

We extend BTA with guarded recursion by adding constants for solutions of guarded recursive specifications and axioms concerning those additional constants. For each guarded recursive specification E and each $X \in \mathrm{V}(E)$, we add a constant standing for the unique solution of E for X to BTA. The constant standing for the unique solution of E for X is denoted by $\langle X|E\rangle$. Moreover, we add the axioms

Table 1. Axioms for guarded recursion

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

Table 2. Approximation induction principle

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

Table 3. Axioms for projection

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

for guarded recursion given in Table 1 to BTA, where we write $\langle t_X|E\rangle$ for t_X with, for all $X \in \mathrm{V}(E)$, all occurrences of X in t_X replaced by $\langle X|E\rangle$. In this table, X, t_X and E stand for an arbitrary variable, an arbitrary term of BTA and an arbitrary guarded recursive specification over BTA, respectively. Side conditions are added to restrict the variables, terms and guarded recursive specifications for which X, t_X and E stand. The equations $\langle X|E\rangle = \langle t_X|E\rangle$ for a fixed E express that the constants $\langle X|E\rangle$ make up a solution of E. The conditional equations $E \Rightarrow X = \langle X|E\rangle$ express that this solution is the only one. RDP stands for recursive definition principle and RSP stands for recursive specification principle.

We often write X for $\langle X|E\rangle$ if E is clear from the context. It should be borne in mind that, in such cases, we use X as a constant.

The projective limit characterization of process equivalence on threads is based on the notion of a finite approximation of depth n. When for all n these approximations are identical for two given threads, both threads are considered identical. This is expressed by the infinitary conditional equation given in Table 2. Following [3, 6], approximation of depth n is phrased in terms of a unary *projection* operator $\pi_n(\_)$. The projection operators are defined inductively by means of equations P0–P3 given in Table 3. In this table, a stands for an arbitrary basic action from $\mathcal{A}$ and n stands for an arbitrary natural number. AIP stands for approximation induction principle.

RDP, RSP and AIP originate from work on ACP [2]. In the setting of ACP, these principles were first formulated in [5]. Like in the setting of ACP, RSP follows from RDP and AIP.

In the structural operational semantics, we represent an execution environment by a function $\rho{:}\mathcal{A}^+ \to \{\mathsf{T}, \mathsf{F}\}$. We write $\mathcal{E}$ for the set of all those functions and $\mathbb{B}$ for the set $\{\mathsf{T}, \mathsf{F}\}$. Given an execution environment ρ and a basic action a, the *derived* execution environment $\frac{\partial}{\partial a}\rho$ is defined by $\frac{\partial}{\partial a}\rho(\alpha) = \rho(\langle a\rangle \frown \alpha)$.[3]

The chosen representation of execution environments is based on the assumption that the reply pro-

[3]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 4. Transition rules of BTA

$$\frac{}{\mathsf{S}{\downarrow}} \qquad \frac{}{\mathsf{D}{\uparrow}}$$

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

$$\frac{x{\downarrow}}{x{\updownarrow}} \qquad \frac{x{\uparrow}}{x{\updownarrow}}$$

duced by an execution environment on completion of the processing of a basic action depends at any stage only on that basic action and the sequence of basic actions processed before. This is a realistic assumption for deterministic execution environments. The representation of execution environments chosen in [13, 7] is suitable for non-deterministic execution environments as well.

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

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

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

- $\langle p, \rho\rangle \xrightarrow{a} \langle p', \rho'\rangle$: in execution environment ρ, thread p can perform action a and after that proceed as thread p' in execution environment ρ';
- $p{\downarrow}$: thread p cannot but terminate successfully;
- $p{\uparrow}$: thread p cannot but become inactive;
- $p{\updownarrow}$: thread p cannot but terminate successfully or become inactive.

The termination or deadlock relation is an auxiliary relation used in the transition rules for cyclic interleaving of threads in Section 10.

The structural operational semantics of BTA is described by the transition rules given in Table 4. The transition rules for the constants for solutions of guarded recursive specifications over BTA are given in Table 5. The transition rules for projection are given in Table 6. In these tables, a stands for an arbitrary basic action from $\mathcal{A}$. In Table 5, X, t_X and E stand for an arbitrary variable, an arbitrary term of BTA and an arbitrary guarded recursive specification over BTA, respectively. In Table 6, n stands for an arbitrary natural number.

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

Table 5. Transition rules for guarded recursion

$\dfrac{\langle\langle t_X \vert E\rangle, \rho\rangle \xrightarrow{a} \langle x', \rho'\rangle}{\langle\langle X \vert E\rangle, \rho\rangle \xrightarrow{a} \langle x', \rho'\rangle}\ X = t_X \in E$	$\dfrac{\langle t_X \vert E\rangle{\downarrow}}{\langle X \vert E\rangle{\downarrow}}\ X = t_X \in E$	$\dfrac{\langle t_X \vert E\rangle{\uparrow}}{\langle X \vert E\rangle{\uparrow}}\ X = t_X \in E$

Table 6. Transition rules for projection

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

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

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

Bisimulation equivalence is a congruence with respect to the postconditional composition operators and the projection operators. This follows immediately from the fact that the transition rules for these operators are in the path format (see e.g. [1]). The axioms given in Tables 1–3 are sound with respect to bisimulation equivalence.

Henceforth, we write $\mathbb{T}_{\mathsf{finrec}}$ for the set of all terms of BTA with guarded recursion in which no constants $\langle X \vert E\rangle$ for infinite E occur, and $\mathcal{T}_{\mathsf{finrec}}$ for the set of all closed terms of BTA with guarded recursion in which no constants $\langle X \vert E\rangle$ for infinite E occur. Moreover, we write $\mathcal{T}_{\mathsf{finrec}}(A)$, where $A \subseteq \mathcal{A}$, for the set of all closed terms from $\mathcal{T}_{\mathsf{finrec}}$ that only contain basic actions from A.

4. Applying Threads to Maurer Machines

In this section, we introduce Maurer machines and add for each Maurer machine H a binary *apply* operator $\_ \bullet_H \_$ to BTA. Moreover, we introduce a notion of computation and related notions in the setting of Maurer machines and BTA. The notions concerned will be used in coming proofs.

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

- $A \subseteq \mathcal{A}$;
- $[\![\_]\!] : A \to (\mathcal{O} \times M)$.

The elements of A are called the *basic actions* of H, and $[\![\_]\!]$ is called the *basic action interpretation function* of H. A and $[\![\_]\!]$ constitute the interface between the Maurer computer and its environment. The basic action interpretation function of H associates with each basic action of H an operation from $\mathcal{O}$ and a memory element from M which are material to the processing of the basic action by H. Let $(O_a, m_a) = [\![a]\!]$ for all $a \in A$. Then the processing of a basic action a by H amount to a state change

Table 7. Defining equations for apply operator

$x \bullet_H \uparrow = \uparrow$	
$\mathsf{S} \bullet_H S = S$	
$\mathsf{D} \bullet_H S = \uparrow$	
$(x \unlhd a \unrhd y) \bullet_H S = x \bullet_H O_a(S)$	if $O_a(S)(m_a) = \mathsf{T}$
$(x \unlhd a \unrhd y) \bullet_H S = y \bullet_H O_a(S)$	if $O_a(S)(m_a) = \mathsf{F}$

Table 8. Rule for divergence

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

according to the operation O_a. In the resulting state, the reply produced by H is contained in memory element m_a.

The apply operators associated with Maurer machines are related to the apply operators introduced in [14]. In applying a thread to a Maurer machine, that Maurer machine is taken as the execution environment of the thread. Applying a thread to a Maurer machine amounts to generating a sequence of state changes according to the operations that the Maurer machine associates with the basic actions performed by the thread. Thus, the apply operators allow for threads to transform states of Maurer machines. Such state transformations produce either a state of the Maurer machine concerned 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 $\_ \bullet_H \_$ associated with Maurer machine $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$ must be a term from $\mathcal{T}_{\mathsf{finrec}}(A)$ and its second argument must be a state from $\mathcal{S} \cup \{\uparrow\}$.

Let $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$ be a Maurer machine, let $p \in \mathcal{T}_{\mathsf{finrec}}(A)$, and let $S \in \mathcal{S}$. Then $p \bullet_H S$ is the state from $\mathcal{S}$ that results if all basic actions performed by thread p are processed by the Maurer machine H beginning in state S. If p is S, then there will be no state change. If p is D, then the result is $\uparrow$.

Let $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$ be a Maurer machine, and let $(O_a, m_a) = [\![a]\!]$ for all $a \in A$. Then the apply operator $\_ \bullet_H \_$ is defined by the equations given in Table 7 and the rule given in Table 8. In these tables, a stands for an arbitrary member of A and S stands for an arbitrary member of $\mathcal{S}$.

Let $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$ be a Maurer machine, let $p \in \mathcal{T}_{\mathsf{finrec}}(A)$, and let $S \in \mathcal{S}$. Then p *converges* from S on H if there exists an $n \in \mathbb{N}$ such that $\pi_n(p) \bullet_H S \neq \uparrow$. We say that p *diverges* from S on H if p does not converge from S on H. The rule from Table 8 can be read as follows: if x diverges from S on H, then $x \bullet_H S$ equals $\uparrow$.

Let $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$ be a Maurer machine, and let $(O_a, m_a) = [\![a]\!]$ for all $a \in A$. Then the *step* relation $\_ \vdash_H \_ \subseteq (\mathcal{T}_{\mathsf{finrec}}(A) \times \mathcal{S}) \times (\mathcal{T}_{\mathsf{finrec}}(A) \times \mathcal{S})$ is inductively defined as follows:

- if $O_a(S)(m_a) = \mathsf{T}$ and $p = p' \unlhd a \unrhd p''$, then $(p, S) \vdash_H (p', O_a(S))$;
- if $O_a(S)(m_a) = \mathsf{F}$ and $p = p' \unlhd a \unrhd p''$, then $(p, S) \vdash_H (p'', O_a(S))$.

In this definition, the occurrence of $=$ in $p = p' \trianglelefteq a \trianglerighteq p''$ stands for provable equality. We have that $(p, S) \vdash_H (p', S')$ implies $p \bullet_H S = p' \bullet_H S'$.

Let $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$ be a Maurer machine. Then a *full* path in $\_ \vdash_H \_$ is one of the following:

- a finite path $\langle (p_0, S_0), \ldots, (p_n, S_n) \rangle$ in $\_ \vdash_H \_$ such that there does not exist a $(p_{n+1}, S_{n+1}) \in \mathcal{T}_{\mathsf{finrec}}(A) \times \mathcal{S}$ with $(p_n, S_n) \vdash_H (p_{n+1}, S_{n+1})$;
- an infinite path $\langle (p_0, S_0), (p_1, S_1), \ldots \rangle$ in $\_ \vdash_H \_$.

Moreover, let $p \in \mathcal{T}_{\mathsf{finrec}}(A)$, and let $S \in \mathcal{S}$. Then the *full path* of (p, S) on H is the unique full path in $\_ \vdash_H \_$ from (p, S). If p converges from S on H, then the full path of (p, S) on H is called the *computation* of (p, S) on H.

Let $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$ be a Maurer machine, and let $p \in \mathcal{T}_{\mathsf{finrec}}(A)$ and $S \in \mathcal{S}$ be such that p converges from S on H. Then we write $\|(p, S)\|_H$ for the least $n \in \mathbb{N}$ such that $\pi_n(p) \bullet_H S \neq \uparrow$. The computation of (p, S) on H is a full path of length $\|(p, S)\|_H$ from (p, S) to $(\mathsf{S}, p \bullet_H S)$. So, although $\|(p, S)\|_H$ is not defined in terms of the computation of (p, S) on H, it is the length of the computation of (p, S) on H.

5. Executing Stored Basic Actions

We enhance Maurer machines step by step till we have attained the result that a single control thread can control the execution on a Maurer machine of any executable finite-state thread stored in the memory of the Maurer machine. In this section, we enhance Maurer machines such that processing of a basic action performed by a thread amounts to first storing it in a special memory element and then executing the operation associated with the basic action stored in that special memory element. However, the thread concerned is not stored in the memory of those Maurer machines. Moreover, storing and executing basic actions cannot be controlled by a single control thread. In Section 7, we enhance Maurer machines further such that storing and executing basic actions can be controlled by a single control thread. In Section 8, we enhance them still further such that they can handle stored threads.

We enhance Maurer machines by extending the memory with a *basic action register* (bar) and a *reply register* (rr), and the operation set with a *store* operation for each action a of the original Maurer machine ($O_{\mathsf{store}:a}$) and an *execute stored basic action* operation (O_{exsba}). Moreover, we replace the basic actions of the original Maurer machine by a basic action store:a for each action a of the original Maurer machine and exsba. Those basic actions are associated with the operations $O_{\mathsf{store}:a}$ and O_{exsba}, respectively. The resulting Maurer machines are called SBA-enhancements. SBA stands for stored basic action.

On the SBA-enhancement of a Maurer machine H, processing of a basic action performed by a thread p amounts to first storing it in the special memory element bar and then executing the operation associated with the basic action stored in bar. For storing basic actions in bar and executing basic actions stored in bar, the special basic actions store:a and exsba are introduced. Thus, processing can be brought under control of a variant of the thread p, viz. the thread obtained by applying the transformation ϕ, which is defined after the precise definition of an SBA-enhancement, to p.

Let $A \subset \mathcal{A}$ be such that for all $a \in A$, store:$a \notin A$. Then it is assumed that store:$a \in \mathcal{A}$ for all $a \in A$. Moreover, it is assumed that exsba $\in \mathcal{A}$.

Let $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$ be a Maurer machine with $\mathsf{bar}, \mathsf{rr} \notin M$, $\mathsf{store}{:}a \notin A$ for all $a \in A$ and $\mathsf{exsba} \notin A$, and let $(O_a, m_a) = [\![a]\!]$ for all $a \in A$. Then the *SBA-enhancement* of H is the Maurer machine $(M', B', \mathcal{S}', \mathcal{O}', A', [\![\_]\!]')$ such that

$$\begin{aligned}
M' &= M \cup \{\mathsf{bar}, \mathsf{rr}\}\ ,\\
B' &= B \cup A \cup \mathbb{B}\ ,\\
\mathcal{S}' &= \{S' : M' \to B' \mid S' \restriction M \in \mathcal{S} \wedge S'(\mathsf{bar}) \in A \wedge S'(\mathsf{rr}) \in \mathbb{B}\}\ ,\\
\mathcal{O}' &= \{O' : \mathcal{S}' \to \mathcal{S}' \mid \exists O \in \mathcal{O} \bullet \forall S' \in \mathcal{S}' \bullet\\
&\qquad (O'(S') \restriction M = O(S' \restriction M) \wedge O'(S') \restriction (M' \setminus M) = S' \restriction (M' \setminus M))\}\\
&\quad \cup \{O_{\mathsf{store}:a} \mid a \in A\} \cup \{O_{\mathsf{exsba}}\}\ ,\\
A' &= \{\mathsf{store}{:}a \mid a \in A\} \cup \{\mathsf{exsba}\}\ ,\\
[\![a]\!]' &= (O_a, \mathsf{rr}) \quad \text{for all } a \in A'\ .
\end{aligned}$$

Here, for each $a \in A$, $O_{\mathsf{store}:a}$ is the unique function from $\mathcal{S}'$ to $\mathcal{S}'$ such that for all $S' \in \mathcal{S}'$:

$$\begin{aligned}
O_{\mathsf{store}:a}(S') \restriction M &= S' \restriction M\ ,\\
O_{\mathsf{store}:a}(S')(\mathsf{bar}) &= a\ ,\\
O_{\mathsf{store}:a}(S')(\mathsf{rr}) &= S'(\mathsf{rr})\ ;
\end{aligned}$$

and O_{exsba} is the unique function from $\mathcal{S}'$ to $\mathcal{S}'$ such that for all $S' \in \mathcal{S}'$:

$$\begin{aligned}
O_{\mathsf{exsba}}(S') \restriction M &= O_{S'(\mathsf{bar})}(S' \restriction M)\ ,\\
O_{\mathsf{exsba}}(S')(\mathsf{bar}) &= S'(\mathsf{bar})\ ,\\
O_{\mathsf{exsba}}(S')(\mathsf{rr}) &= O_{S'(\mathsf{bar})}(S' \restriction M)(m_{S'(\mathsf{bar})})\ .
\end{aligned}$$

Because the memory is extended with only finitely many memory elements and the contents of each of those memory elements is not restricted by the contents of other memory elements, it is easy to check that an SBA-enhancement of a Maurer machine is a Maurer machine indeed. The same remark applies to all subsequent enhancements as well.

The transformation function ϕ on $\mathbb{T}_{\mathsf{finrec}}$ is inductively defined as follows:

$$\begin{aligned}
&\phi(X) = X\ ,\\
&\phi(\mathsf{S}) = \mathsf{S}\ ,\\
&\phi(\mathsf{D}) = \mathsf{D}\ ,\\
&\phi(t_1 \trianglelefteq a \trianglerighteq t_2) = \mathsf{store}{:}a \circ (\phi(t_1) \trianglelefteq \mathsf{exsba} \trianglerighteq \phi(t_2))\ ,\\
&\phi(\langle X_0 | \{X_0 = t_0, \ldots, X_n = t_n\}\rangle) = \langle X_0 | \{X_0 = \phi(t_0), \ldots, X_n = \phi(t_n)\}\rangle\ .
\end{aligned}$$

Applying thread p to a state of Maurer machine H has the same effect as applying the transformation of p to the corresponding state of the SBA-enhancement of H. This is stated rigorously in the following theorem.

Theorem 5.1. (SBA-enhancement)
Let $H' = (M', B', \mathcal{S}', \mathcal{O}', A', [\![\_]\!]')$ be the SBA-enhancement of $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$, let $p \in \mathcal{T}_{\mathsf{finrec}}(A)$, and let $S'_0 \in \mathcal{S}'$. Then $p \bullet_H (S'_0 \restriction M) = (\phi(p) \bullet_{H'} S'_0) \restriction M$.

Proof:
Let $(O_a, m_a) = [\![a]\!]$ for all $a \in A$, and let $(O_a, \mathsf{rr}) = [\![a]\!]'$ for all $a \in A'$. It is easy to see that for all $a \in A$ and $S' \in \mathcal{S}'$:

$$\begin{aligned} O_a(S' \upharpoonright M) &= O_{\mathsf{exsba}}(O_{\mathsf{store}:a}(S')) \upharpoonright M \ , && (1) \\ O_a(S' \upharpoonright M)(m_a) &= O_{\mathsf{exsba}}(O_{\mathsf{store}:a}(S'))(\mathsf{rr}) \ . && (2) \end{aligned}$$

In the case where p converges from $S'_0 \upharpoonright M$ on H, it is easy to prove the theorem by induction on $\|(p, S'_0 \upharpoonright M)\|_H$, using equations (1) and (2). In the case where p diverges from $S'_0 \upharpoonright M$ on H, the theorem follows immediately from the claim that in this case $\pi_n(p) \bullet_H (S'_0 \upharpoonright M) = (\pi_{2n}(\phi(p)) \bullet_{H'} S'_0) \upharpoonright M$ for all $n \in \mathbb{N}$. This claim is easily proved by induction on n, using equations (1) and (2). □

In subsequent sections, we will introduce several other kinds of enhancement of Maurer machines based on the idea that processing of a basic action performed by a thread p amounts to first storing it in a special memory element and then executing the operation associated with the basic action stored in that special memory element. However, for each of those other kinds, processing can be brought under control of a single special thread. This is in most cases accomplished by storing a representation of the thread p in a part of the memory of the enhanced Maurer machine.

6. Representation of Threads

In this section, we make precise how to represent threads in the memory of a Maurer machine.

It is assumed that a fixed but arbitrary finite set $\mathsf{M}_{\mathsf{thr}}$ and a fixed but arbitrary bijection $\mathsf{m}_{\mathsf{thr}} : [0, \mathit{card}(\mathsf{M}_{\mathsf{thr}}) - 1] \to \mathsf{M}_{\mathsf{thr}}$ have been given. $\mathsf{M}_{\mathsf{thr}}$ is called the *thread memory*. We write $\mathit{size}(\mathsf{M}_{\mathsf{thr}})$ for $\mathit{card}(\mathsf{M}_{\mathsf{thr}})$. Let $n, n' \in [0, \mathit{size}(\mathsf{M}_{\mathsf{thr}}) - 1]$ be such that $n \leq n'$. Then, we write $\mathsf{M}_{\mathsf{thr}}[n]$ for $\mathsf{m}_{\mathsf{thr}}(n)$, and $\mathsf{M}_{\mathsf{thr}}[n, n']$ for $\{\mathsf{m}_{\mathsf{thr}}(k) \mid n \leq k \leq n'\}$.

The thread memory is a memory whose elements can be addressed by means of elements of the set $[0, \mathit{size}(\mathsf{M}_{\mathsf{thr}}) - 1]$. We write $\mathsf{MA}_{\mathsf{thr}}$ for $[0, \mathit{size}(\mathsf{M}_{\mathsf{thr}}) - 1]$.

The thread memory elements are meant for containing the representations of nodes that form part of a simple graph representation of a thread. Here, the representation of a node is either S, D or a triple consisting of a basic action and two natural numbers addressing thread memory elements containing representations of other nodes.

Let $n, n' \in \mathsf{MA}_{\mathsf{thr}}$ be such that $n \leq n'$. Then, we write $\mathsf{B}_{\mathsf{thr}}[n, n']$ for $\{\mathsf{S}, \mathsf{D}\} \cup ([n, n'] \times \mathcal{A} \times [n, n'])$. We write $\mathsf{B}_{\mathsf{thr}}$ for $\mathsf{B}_{\mathsf{thr}}[0, \mathit{size}(\mathsf{M}_{\mathsf{thr}}) - 1]$. $\mathsf{B}_{\mathsf{thr}}$ is called the *thread memory base set*. We write $\mathsf{S}_{\mathsf{thr}}$ for the set of all functions $S_{\mathsf{thr}} : \mathsf{M}_{\mathsf{thr}} \to \mathsf{B}_{\mathsf{thr}}$.

Let $p \in \mathcal{T}_{\mathsf{finrec}}$. Then the *set of nodes of the graph representation* of p, written $\mathit{Nodes}(p)$, is the smallest subset of $\mathcal{T}_{\mathsf{finrec}}$ such that:

- $p \in \mathit{Nodes}(p)$;
- if $p' \trianglelefteq a \trianglerighteq q' \in \mathit{Nodes}(p)$, then $p', q' \in \mathit{Nodes}(p)$;
- if $\langle X_0 | \{X_0 = t_0, \ldots, X_n = t_n\} \rangle \in \mathit{Nodes}(p)$ and $\langle t_0 | \{X_0 = t_0, \ldots, X_n = t_n\} \rangle \equiv p' \trianglelefteq a \trianglerighteq q'$, then $p', q' \in \mathit{Nodes}(p)$.

We write $size(p)$ for $card(Nodes(p))$.

It is assumed that for all $p \in \mathcal{T}_{\mathsf{finrec}}$, a fixed but arbitrary bijection $node_p : [0, size(p)-1] \to Nodes(p)$ with $node_p(0) = p$ has been given.

Let $p \in \mathcal{T}_{\mathsf{finrec}}$ be such that $size(p) \leq size(\mathsf{M_{thr}})$. Then the *stored graph representation* of p, written $\mathsf{s_{thr}}(p)$, is the unique function $s_{\mathsf{thr}} : \mathsf{M_{thr}}[0, size(p) - 1] \to \mathsf{B_{thr}}[0, size(p) - 1]$ such that for all $n \in [0, size(p) - 1]$, $s_{\mathsf{thr}}(\mathsf{M_{thr}}[n]) = nrepr_p(node_p(n))$, where the function $nrepr_p : Nodes(p) \to \mathsf{B_{thr}}[0, size(p) - 1]$ is defined as follows:

$$
\begin{aligned}
&nrepr_p(\mathsf{S}) = \mathsf{S}\ ,\\
&nrepr_p(\mathsf{D}) = \mathsf{D}\ ,\\
&nrepr_p(p' \trianglelefteq a \trianglerighteq q') = (node_p{}^{-1}(p'), a, node_p{}^{-1}(q'))\ ,\\
&nrepr_p(\langle X_0 | \{X_0 = t_0, \ldots, X_n = t_n\}\rangle) = nrepr_p(\langle t_0 | \{X_0 = t_0, \ldots, X_n = t_n\}\rangle)\ .
\end{aligned}
$$

We call $\mathsf{s_{thr}}(p)$ a *stored thread*.

Notice that $\mathsf{s_{thr}}(p)$ is not defined for p with $size(p) > size(\mathsf{M_{thr}})$. The size of the thread memory restricts the threads that can be stored.

In [6], program algebra and a hierarchy of program notations for finite-state threads rooted in program algebra are introduced. The lower-level program notations, which are close to existing assembly languages and bring with them test and jump instructions, permit a more efficient stored representation of threads than the one obtained by $\mathsf{s_{thr}}$. In Section 12, we discuss the connection between stored threads and programs in such a program notation.

7. No Stored Threads, but a Single Control Thread

In this section, we enhance Maurer machines such that storing and executing basic actions can be controlled by a single control thread. In Section 8, we enhance them further such that they can handle stored threads as well. The purpose of this section is to demonstrate that control of the execution of any executable finite-state thread by a single control thread is possible without storing the thread to be executed.

We enhance Maurer machines by extending the memory with a *node register* (nr), a *basic action register* (bar) and a *reply register* (rr), and the operation set with a *halt* operation (O_{halt}), two *fetch* operations ($O_{\mathsf{fetch:T}}$, $O_{\mathsf{fetch:F}}$) and an *execute stored basic action* operation (O_{exsba}). Moreover, we replace the basic actions of the original Maurer machine by basic actions halt, fetch:T, fetch:F and exsba, with which the operations O_{halt}, $O_{\mathsf{fetch:T}}$, $O_{\mathsf{fetch:F}}$ and O_{exsba} are associated. The resulting Maurer machines are called SBA′-enhancements.

The node register nr of an SBA′-enhancement for a thread p is meant for containing the number that corresponds to the node of the graph representation of p from which most recently a basic action has been fetched. That node, together with the reply produced on completion of the execution of the basic action concerned, determines the node from which next time a basic action must be fetched. To indicate that no basic action has been fetched yet, nr must initially contain -1. The number corresponding to the node from which the first time a basic action must be fetched, i.e. the root, is 0. The operation $O_{\mathsf{fetch}:r}$ fetches the basic action from thread p that must be executed next if the reply produced on completion of the execution of the last fetched basic action is r. The operation O_{exsba} executes the last fetched basic action. The operation O_{halt} produces the reply T if p terminates successfully after performing the last fetched basic action, and the reply F otherwise.

In the definition of an SBA′-enhancement of a Maurer machine given below, $\mathit{nrepr}_p(n)$, where $n \in [0, \mathit{size}(p) - 1]$, abbreviates $\mathit{nrepr}_p(\mathit{node}_p(n))$.

It is assumed that $\mathsf{halt} \in \mathcal{A}$, that $\mathsf{fetch}{:}r \in \mathcal{A}$ for all $r \in \mathbb{B}$, and that $\mathsf{exsba} \in \mathcal{A}$.

Let $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$ be a Maurer machine with $\mathsf{nr}, \mathsf{bar}, \mathsf{rr} \notin M$, $\mathsf{halt} \notin A$, $\mathsf{fetch}{:}r \notin A$ for all $r \in \mathbb{B}$ and $\mathsf{exsba} \notin A$, and let $(O_a, m_a) = [\![a]\!]$ for all $a \in A$. Let also $p \in \mathcal{T}_{\mathsf{finrec}}(A)$. Then the *SBA′-enhancement* of H for p is the Maurer machine $H' = (M', B', \mathcal{S}', \mathcal{O}', A', [\![\_]\!]')$ such that

$$\begin{aligned}
M' &= M \cup \{\mathsf{nr}, \mathsf{bar}, \mathsf{rr}\}\ ,\\
B' &= B \cup [-1, \mathit{size}(p) - 1] \cup A \cup \mathbb{B}\ ,\\
\mathcal{S}' &= \{S' : M' \to B' \mid S' \restriction M \in \mathcal{S}\ \wedge\\
&\qquad\qquad S'(\mathsf{nr}) \in [-1, \mathit{size}(p) - 1] \wedge S'(\mathsf{bar}) \in A \wedge S'(\mathsf{rr}) \in \mathbb{B}\}\ ,\\
\mathcal{O}' &= \{O' : \mathcal{S}' \to \mathcal{S}' \mid \exists O \in \mathcal{O} \bullet \forall S' \in \mathcal{S}' \bullet\\
&\qquad (O'(S') \restriction M = O(S' \restriction M) \wedge O'(S') \restriction (M' \setminus M) = S' \restriction (M' \setminus M))\}\\
&\quad \cup \{O_{\mathsf{halt}}\} \cup \{O_{\mathsf{fetch}:r} \mid r \in \mathbb{B}\} \cup \{O_{\mathsf{exsba}}\}\ ,\\
A' &= \{\mathsf{halt}\} \cup \{\mathsf{fetch}{:}r \mid r \in \mathbb{B}\} \cup \{\mathsf{exsba}\}\ ,\\
[\![a]\!]' &= (O_a, \mathsf{rr}) \quad \text{for all } a \in A'\ .
\end{aligned}$$

Here, O_{halt} is the unique function from $\mathcal{S}'$ to $\mathcal{S}'$ such that for all $S' \in \mathcal{S}'$:

$$\begin{aligned}
O_{\mathsf{halt}}(S') \restriction M &= S' \restriction M\ ,\\
O_{\mathsf{halt}}(S')(\mathsf{nr}) &= S'(\mathsf{nr})\ ,\\
O_{\mathsf{halt}}(S')(\mathsf{bar}) &= S'(\mathsf{bar})\ ,\\
O_{\mathsf{halt}}(S')(\mathsf{rr}) &= \mathsf{T} && \text{if } \mathit{node}_p(S'(\mathsf{nr})) = \mathsf{S}\ ,\\
O_{\mathsf{halt}}(S')(\mathsf{rr}) &= \mathsf{F} && \text{if } \mathit{node}_p(S'(\mathsf{nr})) \neq \mathsf{S}\ ;
\end{aligned}$$

for each $r \in \mathbb{B}$, $O_{\mathsf{fetch}:r}$ is the unique function from $\mathcal{S}'$ to $\mathcal{S}'$ such that for all $S' \in \mathcal{S}'$:[4]

$$\begin{aligned}
O_{\mathsf{fetch}:r}(S') \restriction M &= S' \restriction M\ ,\\
O_{\mathsf{fetch}:r}(S')(\mathsf{nr}) &= \mathit{nnn}(S', r)\ ,\\
O_{\mathsf{fetch}:r}(S')(\mathsf{bar}) &= \pi_2(\mathit{nrepr}_p(\mathit{nnn}(S', r))) && \text{if } \mathit{node}_p(\mathit{nnn}(S', r)) \notin \{\mathsf{S}, \mathsf{D}\}\ ,\\
O_{\mathsf{fetch}:r}(S')(\mathsf{bar}) &= S'(\mathsf{bar}) && \text{if } \mathit{node}_p(\mathit{nnn}(S', r)) \in \{\mathsf{S}, \mathsf{D}\}\ ,\\
O_{\mathsf{fetch}:r}(S')(\mathsf{rr}) &= \mathsf{T} && \text{if } \mathit{node}_p(\mathit{nnn}(S', r)) \notin \{\mathsf{S}, \mathsf{D}\}\ ,\\
O_{\mathsf{fetch}:r}(S')(\mathsf{rr}) &= \mathsf{F} && \text{if } \mathit{node}_p(\mathit{nnn}(S', r)) \in \{\mathsf{S}, \mathsf{D}\}\ ,
\end{aligned}$$

where $\mathit{nnn} : \mathcal{S}' \times \mathbb{B} \to [0, \mathit{size}(p) - 1]$ is defined as follows:

$$\begin{aligned}
\mathit{nnn}(S', \mathsf{T}) &= \pi_1(\mathit{nrepr}_p(S'(\mathsf{nr}))) && \text{if } S'(\mathsf{nr}) \neq -1 \wedge \mathit{node}_p(S'(\mathsf{nr})) \notin \{\mathsf{S}, \mathsf{D}\}\ ,\\
\mathit{nnn}(S', \mathsf{F}) &= \pi_3(\mathit{nrepr}_p(S'(\mathsf{nr}))) && \text{if } S'(\mathsf{nr}) \neq -1 \wedge \mathit{node}_p(S'(\mathsf{nr})) \notin \{\mathsf{S}, \mathsf{D}\}\ ,\\
\mathit{nnn}(S', r) &= S'(\mathsf{nr}) && \text{if } S'(\mathsf{nr}) \neq -1 \wedge \mathit{node}_p(S'(\mathsf{nr})) \in \{\mathsf{S}, \mathsf{D}\}\ ,\\
\mathit{nnn}(S', r) &= 0 && \text{if } S'(\mathsf{nr}) = -1\ ;
\end{aligned}$$

[4]Holding on to the usual conventions leads to a double use of π_n: it is used as one of the projection operators introduced in Section 3, and it is used to denote the n-th projection function associated with some cartesian product. It is always clear from the context how it is used.

and O_{exsba} is the unique function from $\mathcal{S}'$ to $\mathcal{S}'$ such that for all $S' \in \mathcal{S}'$:

$$\begin{aligned}
O_{\mathsf{exsba}}(S') \restriction M &= O_{S'(\mathsf{bar})}(S' \restriction M) \;, \\
O_{\mathsf{exsba}}(S')(\mathsf{nr}) &= S'(\mathsf{nr}) \;, \\
O_{\mathsf{exsba}}(S')(\mathsf{bar}) &= S'(\mathsf{bar}) \;, \\
O_{\mathsf{exsba}}(S')(\mathsf{rr}) &= O_{S'(\mathsf{bar})}(S' \restriction M)(m_{S'(\mathsf{bar})}) \;.
\end{aligned}$$

To control the execution of a thread, we introduce below a control thread CT. Preceding that, we sketch the behaviour of CT.

CT fetches the next basic action from the thread being executed in accordance with the reply produced on completion of the execution of the last fetched basic action. If that succeeds, then CT first executes that basic action and next returns to fetching the next basic action. Otherwise, the thread being executed has come to an end and CT comes to an end accordingly. In case no basic action has been fetched yet, CT fetches a basic action as if the reply T was produced.

The guarded recursive specification of CT consists of the following equations:

$$\begin{aligned}
CT &= (CT \trianglelefteq \mathsf{exsba} \trianglerighteq CT') \trianglelefteq \mathsf{fetch{:}T} \trianglerighteq (\mathsf{S} \trianglelefteq \mathsf{halt} \trianglerighteq \mathsf{D}) \;, \\
CT' &= (CT \trianglelefteq \mathsf{exsba} \trianglerighteq CT') \trianglelefteq \mathsf{fetch{:}F} \trianglerighteq (\mathsf{S} \trianglelefteq \mathsf{halt} \trianglerighteq \mathsf{D}) \;.
\end{aligned}$$

Applying thread p to a state of Maurer machine H has the same effect as applying control thread CT to the corresponding state of the SBA′-enhancement of H for p. This is stated rigorously in the following theorem.

Theorem 7.1. (SBA′-enhancement)
Let $H' = (M', B', \mathcal{S}', \mathcal{O}', A', [\![\_]\!]')$ be the SBA′-enhancement of $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$ for $p \in \mathcal{T}_{\mathsf{finrec}}(A)$, and let $S'_0 \in \mathcal{S}'$ be such that $S'_0(\mathsf{nr}) = -1$. Then $p \bullet_H (S'_0 \restriction M) = (CT \bullet_{H'} S'_0) \restriction M$.

Proof:
Let $(O_a, m_a) = [\![a]\!]$ for all $a \in A$, and let $(O_a, \mathsf{rr}) = [\![a]\!]'$ for all $a \in A'$. Then it is easy to see that for all $S' \in \mathcal{S}'$ with $node_p(nnn(S', S'(\mathsf{rr}))) \notin \{\mathsf{S}, \mathsf{D}\}$:

$$\begin{aligned}
O_a(S' \restriction M) &= O_{\mathsf{exsba}}(O_{\mathsf{fetch}:r}(S')) \restriction M \;, &(3)\\
O_a(S' \restriction M)(m_a) &= O_{\mathsf{exsba}}(O_{\mathsf{fetch}:r}(S'))(\mathsf{rr}) \;, &(4)
\end{aligned}$$

where $a = \pi_2(nrepr_p(node_p(nnn(S', S'(\mathsf{rr})))))$ and $r = S'(\mathsf{rr})$.

Let (p'_n, S'_n) be the $n{+}1$-th element in the full path of (CT, S'_0) on H'. Then it is easy to prove by induction on n that

$$\begin{aligned}
p'_{2n+2} &= CT && \text{if } S'_{2n+1}(\mathsf{rr}) = \mathsf{T} \wedge S'_{2n+2}(\mathsf{rr}) = \mathsf{T} \;, \\
p'_{2n+2} &= CT' && \text{if } S'_{2n+1}(\mathsf{rr}) = \mathsf{T} \wedge S'_{2n+2}(\mathsf{rr}) = \mathsf{F} \;, \\
p'_{2n+2} &= \mathsf{S} && \text{if } S'_{2n+1}(\mathsf{rr}) = \mathsf{F} \wedge S'_{2n+2}(\mathsf{rr}) = \mathsf{T} \;, \\
p'_{2n+2} &= \mathsf{D} && \text{if } S'_{2n+1}(\mathsf{rr}) = \mathsf{F} \wedge S'_{2n+2}(\mathsf{rr}) = \mathsf{F}
\end{aligned} \qquad (5)$$

(if $2n+2 < \|(CT, S'_0)\|_{H'}$ in case CT converges from S'_0 on H'). Moreover, let (p_n, S_n) be the $n{+}1$-th element in the full path of $(p, S'_0 \restriction M)$ on H. Then, using (3), (4) and (5), it is straightforward to prove by induction on n that:

- p_n is represented by the part of the graph representation of p whose root is $node_p(nnn(S'_{2n}, S'_{2n}(\mathsf{rr})))$;

- $S_n = S'_{2n} \restriction M$

(if $n < \|(p, S'_0 \restriction M)\|_H$ in case p converges from $S'_0 \restriction M$ on H). From this, the theorem follows immediately. $\square$

The SBA′-enhancements of a Maurer machine for different threads have different fetch operations. That is why SBA′-enhancements are inflexible from a practical point of view: it is virtually impossible to change an operation available on a real machine. On the other hand, it is easy to change the stored thread present in the memory of a real machine.

8. Fetching Basic Actions from a Stored Thread

In this section, we enhance Maurer machines such that a single control thread can control the execution on a Maurer machine of any executable finite-state thread stored in the memory of the Maurer machine.

We enhance Maurer machines by extending the memory with a *thread memory* ($\mathsf{M}_{\mathsf{thr}}$), a *thread location register* (tlr), a *basic action register* (bar) and a *reply register* (rr), and the operation set with a *halt* operation (O_{halt}), two *fetch* operations ($O_{\mathsf{fetch:T}}$, $O_{\mathsf{fetch:F}}$) and an *execute stored basic action* operation (O_{exsba}). Moreover, we replace the basic actions of the original Maurer machine by basic actions halt, $\mathsf{fetch:T}$, $\mathsf{fetch:F}$ and exsba, with which the operations O_{halt}, $O_{\mathsf{fetch:T}}$, $O_{\mathsf{fetch:F}}$ and O_{exsba} are associated. The resulting Maurer machines are called ST-4O-enhancements. ST stands for stored thread and 4O indicates that there are four control operations available. The operations associated with basic actions halt, $\mathsf{fetch:T}$, $\mathsf{fetch:F}$ and exsba in ST-4O-enhancements differ from the operations associated with those basic actions in SBA′-enhancements.

The thread location register tlr is meant for containing the address of the thread memory element from which most recently a basic action has been fetched. The contents of that thread memory element, together with the reply produced on completion of the execution of the basic action concerned, determines the thread memory element from which next time a basic action must be fetched. To indicate that no basic action has been fetched yet, tlr must initially contain -1. The thread memory element from which the first time a basic action must be fetched is the one at address 0. For a given thread p, the operations O_{halt}, $O_{\mathsf{fetch:T}}$, $O_{\mathsf{fetch:F}}$ and O_{exsba} have essentially the same effect on an ST-4O-enhancement of a Maurer machine and an SBA′-enhancement of the same Maurer machine for p if the thread memory of the ST-4O-enhancement contains $\mathsf{s}_{\mathsf{thr}}(p)$. The main difference is that the effects of $O_{\mathsf{fetch:T}}$ and $O_{\mathsf{fetch:F}}$ on the ST-4O-enhancement are obtained by actually fetching basic actions from a stored graph representation of p in its thread memory, whereas on the SBA′-enhancement the effects that look to be obtained by fetching are fully embedded in the operations.

Let $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$ be a Maurer machine with $\mathsf{M}_{\mathsf{thr}} \not\subseteq M$, $\mathsf{tlr}, \mathsf{bar}, \mathsf{rr} \notin M$, $\mathsf{halt} \notin A$, $\mathsf{fetch}{:}r \notin A$ for all $r \in \mathbb{B}$ and $\mathsf{exsba} \notin A$, and let $(O_a, m_a) = [\![a]\!]$ for all $a \in A$. Then the *ST-4O-*

enhancement of H is the Maurer machine $H' = (M', B', \mathcal{S}', \mathcal{O}', A', [\![\_]\!]')$ such that

$$\begin{aligned}
M' &= M \cup \mathsf{M_{thr}} \cup \{\mathsf{tlr}, \mathsf{bar}, \mathsf{rr}\} \;, \\
B' &= B \cup \mathsf{B_{thr}} \cup \mathsf{MA_{thr}} \cup \{-1\} \cup A \cup \mathbb{B} \;, \\
\mathcal{S}' &= \{S' : M' \to B' \mid S' \restriction M \in \mathcal{S} \wedge S' \restriction \mathsf{M_{thr}} \in \mathsf{S_{thr}} \wedge \\
&\qquad S'(\mathsf{tlr}) \in \mathsf{MA_{thr}} \cup \{-1\} \wedge S'(\mathsf{bar}) \in A \wedge S'(\mathsf{rr}) \in \mathbb{B}\} \;, \\
\mathcal{O}' &= \{O' : \mathcal{S}' \to \mathcal{S}' \mid \exists O \in \mathcal{O} \bullet \forall S' \in \mathcal{S}' \bullet \\
&\qquad (O'(S') \restriction M = O(S' \restriction M) \wedge O'(S') \restriction (M' \setminus M) = S' \restriction (M' \setminus M))\} \\
&\qquad \cup \{O_{\mathsf{halt}}\} \cup \{O_{\mathsf{fetch}:r} \mid r \in \mathbb{B}\} \cup \{O_{\mathsf{exsba}}\} \;, \\
A' &= \{\mathsf{halt}\} \cup \{\mathsf{fetch}{:}r \mid r \in \mathbb{B}\} \cup \{\mathsf{exsba}\} \;, \\
[\![a]\!]' &= (O_a, \mathsf{rr}) \quad \text{for all } a \in A' \;.
\end{aligned}$$

Here, O_{halt} is the unique function from $\mathcal{S}'$ to $\mathcal{S}'$ such that for all $S' \in \mathcal{S}'$:

$$\begin{aligned}
O_{\mathsf{halt}}(S') \restriction M &= S' \restriction M \;, \\
O_{\mathsf{halt}}(S') \restriction \mathsf{M_{thr}} &= S' \restriction \mathsf{M_{thr}} \;, \\
O_{\mathsf{halt}}(S')(\mathsf{tlr}) &= S'(\mathsf{tlr}) \;, \\
O_{\mathsf{halt}}(S')(\mathsf{bar}) &= S'(\mathsf{bar}) \;, \\
O_{\mathsf{halt}}(S')(\mathsf{rr}) &= \mathsf{T} && \text{if } S'(\mathsf{M_{thr}}[S'(\mathsf{tlr})]) = \mathsf{S} \;, \\
O_{\mathsf{halt}}(S')(\mathsf{rr}) &= \mathsf{F} && \text{if } S'(\mathsf{M_{thr}}[S'(\mathsf{tlr})]) \neq \mathsf{S} \;;
\end{aligned}$$

for each $r \in \mathbb{B}$, $O_{\mathsf{fetch}:r}$ is the unique function from $\mathcal{S}'$ to $\mathcal{S}'$ such that for all $S' \in \mathcal{S}'$:

$$\begin{aligned}
O_{\mathsf{fetch}:r}(S') \restriction M &= S' \restriction M \;, \\
O_{\mathsf{fetch}:r}(S') \restriction \mathsf{M_{thr}} &= S' \restriction \mathsf{M_{thr}} \;, \\
O_{\mathsf{fetch}:r}(S')(\mathsf{tlr}) &= ntla(S', r) \;, \\
O_{\mathsf{fetch}:r}(S')(\mathsf{bar}) &= \pi_2(S'(\mathsf{M_{thr}}[ntla(S', r)])) && \text{if } S'(\mathsf{M_{thr}}[ntla(S', r)]) \notin \{\mathsf{S}, \mathsf{D}\} \;, \\
O_{\mathsf{fetch}:r}(S')(\mathsf{bar}) &= S'(\mathsf{bar}) && \text{if } S'(\mathsf{M_{thr}}[ntla(S', r)]) \in \{\mathsf{S}, \mathsf{D}\} \;, \\
O_{\mathsf{fetch}:r}(S')(\mathsf{rr}) &= \mathsf{T} && \text{if } S'(\mathsf{M_{thr}}[ntla(S', r)]) \notin \{\mathsf{S}, \mathsf{D}\} \;, \\
O_{\mathsf{fetch}:r}(S')(\mathsf{rr}) &= \mathsf{F} && \text{if } S'(\mathsf{M_{thr}}[ntla(S', r)]) \in \{\mathsf{S}, \mathsf{D}\} \;,
\end{aligned}$$

where $ntla : \mathcal{S}' \times \mathbb{B} \to \mathsf{MA_{thr}}$ is defined as follows:

$$\begin{aligned}
ntla(S', \mathsf{T}) &= \pi_1(S'(\mathsf{M_{thr}}[S'(\mathsf{tlr})])) && \text{if } S'(\mathsf{tlr}) \in \mathsf{MA_{thr}} \wedge S'(\mathsf{M_{thr}}[S'(\mathsf{tlr})]) \notin \{\mathsf{S}, \mathsf{D}\} \;, \\
ntla(S', \mathsf{F}) &= \pi_3(S'(\mathsf{M_{thr}}[S'(\mathsf{tlr})])) && \text{if } S'(\mathsf{tlr}) \in \mathsf{MA_{thr}} \wedge S'(\mathsf{M_{thr}}[S'(\mathsf{tlr})]) \notin \{\mathsf{S}, \mathsf{D}\} \;, \\
ntla(S', r) &= S'(\mathsf{tlr}) && \text{if } S'(\mathsf{tlr}) \in \mathsf{MA_{thr}} \wedge S'(\mathsf{M_{thr}}[S'(\mathsf{tlr})]) \in \{\mathsf{S}, \mathsf{D}\} \;, \\
ntla(S', r) &= 0 && \text{if } S'(\mathsf{tlr}) \notin \mathsf{MA_{thr}} \;;
\end{aligned}$$

and O_{exsba} is the unique function from $\mathcal{S}'$ to $\mathcal{S}'$ such that for all $S' \in \mathcal{S}'$:

$$\begin{aligned}
O_{\mathsf{exsba}}(S') \restriction M &= O_{S'(\mathsf{bar})}(S' \restriction M) \;, \\
O_{\mathsf{exsba}}(S') \restriction \mathsf{M_{thr}} &= S' \restriction \mathsf{M_{thr}} \;, \\
O_{\mathsf{exsba}}(S')(\mathsf{tlr}) &= S'(\mathsf{tlr}) \;, \\
O_{\mathsf{exsba}}(S')(\mathsf{bar}) &= S'(\mathsf{bar}) \;, \\
O_{\mathsf{exsba}}(S')(\mathsf{rr}) &= O_{S'(\mathsf{bar})}(S' \restriction M)(m_{S'(\mathsf{bar})}) \;.
\end{aligned}$$

Consider again the guarded recursive specification over BTA that consists of the following equations:

$$\begin{aligned} CT &= (CT \trianglelefteq \mathsf{exsba} \trianglerighteq CT') \trianglelefteq \mathsf{fetch{:}T} \trianglerighteq (\mathsf{S} \trianglelefteq \mathsf{halt} \trianglerighteq \mathsf{D})\ , \\ CT' &= (CT \trianglelefteq \mathsf{exsba} \trianglerighteq CT') \trianglelefteq \mathsf{fetch{:}F} \trianglerighteq (\mathsf{S} \trianglelefteq \mathsf{halt} \trianglerighteq \mathsf{D})\ . \end{aligned}$$

Applying thread p to a state of Maurer machine H has the same effect as applying control thread CT to the corresponding state of the ST-4O-enhancement of H in which the thread memory contains the stored graph representation of p. This is stated rigorously in the following theorem.

Theorem 8.1. (ST-4O-enhancement)
Let $H' = (M', B', \mathcal{S}', \mathcal{O}', A', [\![\_]\!]')$ be the ST-4O-enhancement of $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$, let $p \in \mathcal{T}_{\mathsf{finrec}}(A)$ be such that $size(p) \leq size(\mathsf{M_{thr}})$, and let $S'_0 \in \mathcal{S}'$ be such that $S'_0 \restriction \mathsf{M_{thr}}[0, size(p) - 1] = \mathsf{s_{thr}}(p)$ and $S'_0(\mathsf{tlr}) = -1$. Then $p \bullet_H (S'_0 \restriction M) = (CT \bullet_{H'} S'_0) \restriction M$.

Proof:
Let $(O_a, m_a) = [\![a]\!]$ for all $a \in A$, and let $(O_a, \mathsf{rr}) = [\![a]\!]'$ for all $a \in A'$. Then it is easy to see that for all $S' \in \mathcal{S}'$ with $S'(\mathsf{M_{thr}}[ntla(S', S'(\mathsf{rr}))]) \notin \{\mathsf{S}, \mathsf{D}\}$:

$$O_a(S' \restriction M) = O_{\mathsf{exsba}}(O_{\mathsf{fetch}:r}(S')) \restriction M\ , \tag{6}$$

$$O_a(S' \restriction M)(m_a) = O_{\mathsf{exsba}}(O_{\mathsf{fetch}:r}(S'))(\mathsf{rr})\ , \tag{7}$$

where $a = \pi_2(S'(\mathsf{M_{thr}}[ntla(S', S'(\mathsf{rr}))]))$ and $r = S'(\mathsf{rr})$.

Let (p'_n, S'_n) be the $n{+}1$-th element in the full path of (CT, S'_0) on H'. Then it is easy to prove by induction on n that

$$\begin{aligned} p'_{2n+2} &= CT && \text{if } S'_{2n+1}(\mathsf{rr}) = \mathsf{T} \wedge S'_{2n+2}(\mathsf{rr}) = \mathsf{T}\ , \\ p'_{2n+2} &= CT' && \text{if } S'_{2n+1}(\mathsf{rr}) = \mathsf{T} \wedge S'_{2n+2}(\mathsf{rr}) = \mathsf{F}\ , \\ p'_{2n+2} &= \mathsf{S} && \text{if } S'_{2n+1}(\mathsf{rr}) = \mathsf{F} \wedge S'_{2n+2}(\mathsf{rr}) = \mathsf{T}\ , \\ p'_{2n+2} &= \mathsf{D} && \text{if } S'_{2n+1}(\mathsf{rr}) = \mathsf{F} \wedge S'_{2n+2}(\mathsf{rr}) = \mathsf{F} \end{aligned} \tag{8}$$

(if $2n + 2 < \|(CT, S'_0)\|_{H'}$ in case CT converges from S'_0 on H'). Moreover, let (p_n, S_n) be the $n{+}1$-th element in the full path of $(p, S'_0 \restriction M)$ on H. Then, using (6), (7) and (8), it is straightforward to prove by induction on n that:

- p_n is represented by the part of $\mathsf{s_{thr}}(p)$ to which $ntla(S'_{2n}, S'_{2n}(\mathsf{rr}))$ points;
- $S_n = S'_{2n} \restriction M$

(if $n < \|(p, S'_0 \restriction M)\|_H$ in case p converges from $S'_0 \restriction M$ on H). From this, the theorem follows immediately. □

Notice that the proof of Theorem 7.1 and the proof of Theorem 8.1 follow similar lines.

The size of a stored thread may exceed the size of the thread memory of an ST-4O-enhancement. In other words, an ST-4O-enhancement cannot handle finite-state threads of arbitrary size. Section 11 shows how to get around this limitation.

9. A Universal Control operation

On an ST-4O-enhancement of a Maurer machine, four operations are available for controlling the execution of any finite-state thread stored in the memory of the Maurer machine by means of a single control thread. In this section, we introduce ST-1O-enhancements, which have a single universal control operation available for that purpose.

We enhance Maurer machines by extending the memory with a *thread memory* ($\mathsf{M_{thr}}$), a *thread location register* (tlr), a *basic action register* (bar), a *reply register* (rr) and a *fetch mode register* (fmr), and the operation set with a *step* operation (O_{step}). Moreover, we replace the basic actions of the original Maurer machine by one basic action, step, with which the operation O_{step} is associated. The resulting Maurer machines are called ST-1O-enhancements. ST stands again for stored thread and 1O indicates that there is one control operation available.

Consecutive executions of the operation O_{step} alternate between a fetch mode and an execute mode. The fetch mode register fmr is meant for containing a flag that indicates whether the next time step is executed the mode is fetch mode. The contents of that register, together with the contents of the reply register, determines whether the next time O_{step} is executed actually O_{halt}, $O_{\mathsf{fetch}}{:}\mathsf{T}$, $O_{\mathsf{fetch}}{:}\mathsf{F}$ or O_{exsba} is executed.

It is assumed that $\mathsf{step} \in \mathcal{A}$.

Let $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$ be a Maurer machine with $\mathsf{M_{thr}} \not\subseteq M$, $\mathsf{tlr}, \mathsf{bar}, \mathsf{rr}, \mathsf{fmr} \notin M$ and $\mathsf{step} \notin A$, and let $(O_a, m_a) = [\![a]\!]$ for all $a \in A$. Then the *ST-1O-enhancement* of H is the Maurer machine $H' = (M', B', \mathcal{S}', \mathcal{O}', A', [\![\_]\!]')$ such that

$$\begin{aligned}
M' &= M \cup \mathsf{M_{thr}} \cup \{\mathsf{tlr}, \mathsf{bar}, \mathsf{rr}, \mathsf{fmr}\}\ ,\\
B' &= B \cup \mathsf{B_{thr}} \cup \mathsf{MA_{thr}} \cup \{-1\} \cup A \cup \mathbb{B}\ ,\\
\mathcal{S}' &= \{S' : M' \to B' \mid S' \restriction M \in \mathcal{S} \land S' \restriction \mathsf{M_{thr}} \in \mathsf{S_{thr}} \land\\
&\qquad S'(\mathsf{tlr}) \in \mathsf{MA_{thr}} \cup \{-1\} \land S'(\mathsf{bar}) \in A \land S'(\mathsf{rr}) \in \mathbb{B} \land S'(\mathsf{fmr}) \in \mathbb{B}\}\ ,\\
\mathcal{O}' &= \{O' : \mathcal{S}' \to \mathcal{S}' \mid \exists O \in \mathcal{O} \bullet \forall S' \in \mathcal{S}' \bullet\\
&\qquad (O'(S') \restriction M = O(S' \restriction M) \land O'(S') \restriction (M' \setminus M) = S' \restriction (M' \setminus M))\}\\
&\quad \cup \{O_{\mathsf{step}}\}\ ,\\
A' &= \{\mathsf{step}\}\ ,\\
[\![\mathsf{step}]\!]' &= (O_{\mathsf{step}}, \mathsf{rr})\ .
\end{aligned}$$

Here, O_{step} is the unique function from $\mathcal{S}'$ to $\mathcal{S}'$ such that for all $S' \in \mathcal{S}'$:

$$\begin{aligned}
&O_{\mathsf{step}}(S') \restriction M'' = O_{\mathsf{fetch}:r}(S' \restriction M'') && \text{if } S'(\mathsf{fmr}) = \mathsf{T} \land S'(\mathsf{rr}) = r\ ,\\
&O_{\mathsf{step}}(S') \restriction M'' = O_{\mathsf{exsba}}(S' \restriction M'') && \text{if } S'(\mathsf{fmr}) = \mathsf{F} \land S'(\mathsf{rr}) = \mathsf{T}\ ,\\
&O_{\mathsf{step}}(S') \restriction M'' = O_{\mathsf{halt}}(S' \restriction M'') && \text{if } S'(\mathsf{fmr}) = \mathsf{F} \land S'(\mathsf{rr}) = \mathsf{F}\ ,\\
&O_{\mathsf{step}}(S')(\mathsf{fmr}) = \mathsf{F} && \text{if } S'(\mathsf{fmr}) = \mathsf{T}\ ,\\
&O_{\mathsf{step}}(S')(\mathsf{fmr}) = \mathsf{T} && \text{if } S'(\mathsf{fmr}) = \mathsf{F}\ ,
\end{aligned}$$

where $M'' = M \cup \mathsf{M_{thr}} \cup \{\mathsf{tlr}, \mathsf{bar}, \mathsf{rr}\}$ and $O_{\mathsf{fetch}:r}$, O_{exsba} and O_{halt} are defined as in the definition of the ST-4O-enhancement.

To control the execution of a thread, we introduce below a control thread CT''. Preceding that, we sketch the behaviour of CT''.

CT'' is reminiscent of CT. An odd step of CT'' is actually a fetch step, which may fail because of termination or deadlock of the controlled thread. An even step of CT'' is actually an execute step if the preceding fetch step did not fail. Otherwise, it is a halt step. In a fetch step, the next basic action from the controlled thread is fetched in accordance with the reply produced on completion of the execution of the last fetched basic action by inspecting the reply register.

The guarded recursive specification of CT'' consists of the following equation:

$$CT'' = (\mathsf{step} \circ CT'') \trianglelefteq \mathsf{step} \trianglerighteq (\mathsf{S} \trianglelefteq \mathsf{step} \trianglerighteq \mathsf{D}) \; .$$

Applying thread p to a state of Maurer machine H has the same effect as applying control thread CT'' to the corresponding state of the ST-1O-enhancement of H in which the thread memory contains the stored graph representation of p. This is stated rigorously in the following theorem.

Theorem 9.1. (ST-1O-enhancement)
Let $H' = (M', B', \mathcal{S}', \mathcal{O}', A', [\![\_]\!]')$ be the ST-1O-enhancement of $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$, let $p \in \mathcal{T}_{\mathsf{finrec}}(A)$ be such that $size(p) \leq size(\mathsf{M_{thr}})$, and let $S'_0 \in \mathcal{S}'$ be such that $S'_0 \restriction \mathsf{M_{thr}}[0, size(p) - 1] = \mathsf{s_{thr}}(p)$, $S'_0(\mathsf{tlr}) = -1$ and $S'_0(\mathsf{fmr}) = \mathsf{T}$. Then $p \bullet_H (S'_0 \restriction M) = (CT'' \bullet_{H'} S'_0) \restriction M$.

Proof:
The proof follows the same line as the proof of Theorem 8.1. In the proof, the equations corresponding to equations (6) and (7) hold only for states S' with $S'(\mathsf{fmr}) = \mathsf{T}$. This does not stand in the way of following the same line, because this extra condition is satisfied by all states S' that have to be related to the state component of an element in the full path of $(p, S'_0 \restriction M)$ on H. □

10. Parallel Maurer Machines and Interleaving of Threads

In Section 11, we will show that a Maurer machine with a fixed finite memory can deal with any finite-state thread, provided that it is put in parallel with a Maurer machine of a suitable kind that can hold the thread concerned. In this section, we introduce the parallel composition of Maurer machines. Moreover, because the control threads of the Maurer machines have to be interleaved if they are put in parallel, we add an operator for that purpose to BTA.

Let $H_i = (M_i, B_i, \mathcal{S}_i, \mathcal{O}_i, A_i, [\![\_]\!]_i)$, for $i = 1, 2$, be Maurer machines with for all $x \in M_1 \cap M_2$ either $\forall O_1 \in \mathcal{O}_1 \bullet x \notin OR(O_1)$ or $\forall O_2 \in \mathcal{O}_2 \bullet x \notin OR(O_2)$, and $A_1 \cap A_2 = \emptyset$. Then the *parallel composition* of H_1 and H_2, written $H_1 \parallel H_2$, is the unique Maurer machine $(M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$ such that

$$\begin{aligned}
M &= M_1 \cup M_2 \; , \\
B &= B_1 \cup B_2 \; , \\
\mathcal{S} &= \{S : M \to B \mid S \restriction M_1 \in \mathcal{S}_1 \wedge S \restriction M_2 \in \mathcal{S}_2\} \; , \\
\mathcal{O} &= \mathcal{O}_1 \cup \mathcal{O}_2 \; , \\
A &= A_1 \cup A_2 \; , \\
[\![a]\!] &= [\![a]\!]_1 \quad \text{if } a \in A_1 \; , \\
[\![a]\!] &= [\![a]\!]_2 \quad \text{if } a \in A_2 \; .
\end{aligned}$$

Note that the parallel composition of two Maurer machines is defined only if each common memory element is read-only for at least one of the Maurer machines. It is usual that the common memory

Table 9. Axioms for cyclic interleaving

$\|(\langle\,\rangle) = \mathsf{S}$	CSI1
$\|(\langle \mathsf{S}\rangle \frown \alpha) = \|(\alpha)$	CSI2
$\|(\langle \mathsf{D}\rangle \frown \alpha) = \mathsf{S}_\mathsf{D}(\|(\alpha))$	CSI3
$\|(\langle x \trianglelefteq a \trianglerighteq y\rangle \frown \alpha) = \|(\alpha \frown \langle x\rangle) \trianglelefteq a \trianglerighteq \|(\alpha \frown \langle y\rangle)$	CSI4

Table 10. Axioms for deadlock at termination

$\mathsf{S}_\mathsf{D}(\mathsf{S}) = \mathsf{D}$	S2D1
$\mathsf{S}_\mathsf{D}(\mathsf{D}) = \mathsf{D}$	S2D2
$\mathsf{S}_\mathsf{D}(x \trianglelefteq a \trianglerighteq y) = \mathsf{S}_\mathsf{D}(x) \trianglelefteq a \trianglerighteq \mathsf{S}_\mathsf{D}(y)$	S2D3

elements do duty for communication between the parallel Maurer machines. The parallel composition of Maurer machines is not considered in [19, 20].

It is assumed that a collection of threads to be interleaved takes the form of a sequence of threads, called a *thread vector*. Strategic interleaving operators turn a thread vector of arbitrary length into a single thread. This single thread obtained via a strategic interleaving operator is also called a *multi-thread*. Formally, however multi-threads are threads as well.

In this section, we only cover the simplest interleaving strategy, namely *cyclic interleaving*. Cyclic interleaving basically operates as follows: at each stage of the interleaving, the first thread in the thread vector gets a turn to perform a basic action and then the thread vector undergoes a cyclic permutation. We mean by a cyclic permutation of a thread vector that the first thread in the thread vector becomes the last one and all others move one position to the left. If one thread in the thread vector deadlocks, the whole does not deadlock till all others have terminated or deadlocked. An important property of cyclic interleaving is that it is fair, i.e. there will always come a next turn for all active threads. Other plausible interleaving strategies are treated in [13]. The strategic interleaving operator for cyclic interleaving is denoted by $\|(\_)$.

The axioms for cyclic interleaving are given in Table 9. In CSI3, the auxiliary *deadlock at termination* operator $\mathsf{S}_\mathsf{D}(\_)$ is used. It turns termination into deadlock. Its axioms appear in Table 10. In these tables, a stands for an arbitrary basic action from $\mathcal{A}$.

The structural operational semantics of BTA extended with cyclic interleaving is described by the transition rules given in Tables 4 and 11. In these tables, a stands for an arbitrary basic action from $\mathcal{A}$. Without the termination or deadlock relation $\_\updownarrow$, we would need negative premises in the second, fourth and sixth transition rule.

Bisimulation equivalence is also a congruence with respect to the cyclic interleaving operator and the deadlock at termination operator. This follows immediately from the fact that the transition rules from Tables 4 and 11 constitute a complete transition system specification in relaxed panth format (see e.g. [21]). The axioms given in Tables 9 and 10 are sound with respect to bisimulation equivalence.

Table 11. Transition rules for cyclic interleaving and deadlock at termination

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

$$\frac{x_1{\updownarrow},\ldots,x_k{\updownarrow},x_l{\uparrow},\langle x_{k+1},\rho\rangle \xrightarrow{a} \langle x'_{k+1},\rho'\rangle}{\langle \|(\langle x_1\rangle \curvearrowright \ldots \curvearrowright \langle x_{k+1}\rangle \curvearrowright \alpha),\rho\rangle \xrightarrow{a} \langle \|(\alpha \curvearrowright \langle \mathsf{D}\rangle \curvearrowright \langle x'_{k+1}\rangle),\rho'\rangle} \quad (k \geq l > 0)$$

$$\frac{x_1{\downarrow},\ldots,x_k{\downarrow}}{\|(\langle x_1\rangle \curvearrowright \ldots \curvearrowright \langle x_k\rangle){\downarrow}} \qquad \frac{x_1{\updownarrow},\ldots,x_k{\updownarrow},x_l{\uparrow}}{\|(\langle x_1\rangle \curvearrowright \ldots \curvearrowright \langle x_k\rangle){\uparrow}} \quad (k \geq l > 0)$$

$$\frac{\langle x,\rho\rangle \xrightarrow{a} \langle x',\rho'\rangle}{\langle \mathsf{S_D}(x),\rho\rangle \xrightarrow{a} \langle \mathsf{S_D}(x'),\rho'\rangle} \qquad \frac{x{\updownarrow}}{\mathsf{S_D}(x){\uparrow}}$$

11. Dealing with Finite-State Threads of Arbitrary Size

In this section, we show that finite-state threads of arbitrary size can be dealt with by means of an enhanced Maurer machine that does the execution of stored basic actions, but leaves the fetching of those basic actions to a remote Maurer machine whose memory size is sufficient for the thread concerned.

We enhance Maurer machines by extending the memory with a *basic action register* (bar), a *reply register* (rr), a *remote reply register* (rrr) and a *stop mode register* (smr), and the operation set with a *halt* operation (O_{halt}) and an *execute stored basic action* operation (O_{exsba}). Moreover, we replace the basic actions of the original Maurer machine by basic actions halt and exsba, with which the operations O_{halt} and O_{exsba} are associated. The resulting Maurer machines are called RST-enhancements. RST stands for remote stored thread.

We also introduce a Maurer machine with a memory consisting of a *thread memory* ($\mathsf{M_{thr}}$), a *thread location register* (tlr), a *basic action register* (bar), a *reply register* (rr), a *remote reply register* (rrr), and a *stop mode register* (smr), and an operation set consisting of a *fetch* operation (O_{fetch}). Moreover, this Maurer machine has one basic action, fetch, with which the operation O_{fetch} is associated. The resulting Maurer machine is called the remote machine for stored threads.

The common memory elements of the RST-enhancement H' of a Maurer machine and the remote machine H'' for a stored thread are $\mathsf{bar}, \mathsf{rr}, \mathsf{rrr}, \mathsf{smr}$. The memory elements $\mathsf{bar}, \mathsf{rrr}, \mathsf{smr}$ are not changed by any operations of H' and the memory element rr is not changed by any operations of H''. So, the parallel composition $H' \parallel H''$ is defined (cf. Section 10). The fetch, execute and halt operations found here are similar to the ones of an ST-4O-enhancement. The operation fetch has the same effect as either fetch:T or fetch:F depending on the contents of rr. The operation exsba has no effect if rrr contains F.

Let $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$ be a Maurer machine with $\mathsf{M_{thr}} \not\subseteq M$, $\mathsf{tlr}, \mathsf{bar}, \mathsf{rr}, \mathsf{rrr}, \mathsf{smr} \notin M$, $\mathsf{halt} \notin A$, $\mathsf{fetch}{:}r \notin A$ for all $r \in \mathbb{B}$ and $\mathsf{exsba} \notin A$, and let $(O_a, m_a) = [\![a]\!]$ for all $a \in A$. Then the *RST-enhancement* of H is the Maurer machine $H' = (M', B', \mathcal{S}', \mathcal{O}', A', [\![\_]\!]')$ such that

$$
\begin{aligned}
M' &= M \cup \{\mathsf{bar}, \mathsf{rr}, \mathsf{rrr}, \mathsf{smr}\}\ , \\
B' &= B \cup A \cup \mathbb{B}\ , \\
\mathcal{S}' &= \{S' : M' \to B' \mid S' \restriction M \in \mathcal{S} \wedge \\
&\qquad\qquad S'(\mathsf{bar}) \in A \wedge S'(\mathsf{rr}) \in \mathbb{B} \wedge S'(\mathsf{rrr}) \in \mathbb{B} \wedge S'(\mathsf{smr}) \in \mathbb{B}\}\ , \\
\mathcal{O}' &= \{O' : \mathcal{S}' \to \mathcal{S}' \mid \exists O \in \mathcal{O} \bullet \forall S' \in \mathcal{S}' \bullet \\
&\qquad (O'(S') \restriction M = O(S' \restriction M) \wedge O'(S') \restriction (M' \setminus M) = S' \restriction (M' \setminus M))\} \\
&\qquad \cup \{O_{\mathsf{halt}}, O_{\mathsf{exsba}}\}\ , \\
A' &= \{\mathsf{halt}, \mathsf{exsba}\}\ , \\
[\![\mathsf{halt}]\!]' &= (O_{\mathsf{halt}}, \mathsf{rr})\ , \\
[\![\mathsf{exsba}]\!]' &= (O_{\mathsf{exsba}}, \mathsf{rrr})\ .
\end{aligned}
$$

Here, O_{halt} is the unique function from $\mathcal{S}'$ to $\mathcal{S}'$ such that for all $S' \in \mathcal{S}'$:

$$
\begin{aligned}
O_{\mathsf{halt}}(S') \restriction M &= S' \restriction M\ , \\
O_{\mathsf{halt}}(S')(\mathsf{bar}) &= S'(\mathsf{bar})\ , \\
O_{\mathsf{halt}}(S')(\mathsf{rr}) &= S'(\mathsf{smr})\ , \\
O_{\mathsf{halt}}(S')(\mathsf{rrr}) &= S'(\mathsf{rrr})\ , \\
O_{\mathsf{halt}}(S')(\mathsf{smr}) &= S'(\mathsf{smr})\ ;
\end{aligned}
$$

and O_{exsba} is the unique function from $\mathcal{S}'$ to $\mathcal{S}'$ such that for all $S' \in \mathcal{S}'$:

$$
\begin{aligned}
O_{\mathsf{exsba}}(S') \restriction M &= O_{S'(\mathsf{bar})}(S' \restriction M) && \text{if } S'(\mathsf{rrr}) = \mathsf{T}\ , \\
O_{\mathsf{exsba}}(S') \restriction M &= S' \restriction M && \text{if } S'(\mathsf{rrr}) = \mathsf{F}\ , \\
O_{\mathsf{exsba}}(S')(\mathsf{bar}) &= S'(\mathsf{bar})\ , \\
O_{\mathsf{exsba}}(S')(\mathsf{rr}) &= O_{S'(\mathsf{bar})}(S' \restriction M)(m_{S'(\mathsf{bar})}) && \text{if } S'(\mathsf{rrr}) = \mathsf{T}\ , \\
O_{\mathsf{exsba}}(S')(\mathsf{rr}) &= S'(\mathsf{rr}) && \text{if } S'(\mathsf{rrr}) = \mathsf{F}\ , \\
O_{\mathsf{exsba}}(S')(\mathsf{rrr}) &= S'(\mathsf{rrr})\ , \\
O_{\mathsf{exsba}}(S')(\mathsf{smr}) &= S'(\mathsf{smr})\ .
\end{aligned}
$$

The *remote machine for stored threads* is the Maurer machine $H'' = (M'', B'', \mathcal{S}'', \mathcal{O}'', A'', [\![\_]\!]'')$ such that

$$
\begin{aligned}
M'' &= \mathsf{M_{thr}} \cup \{\mathsf{tlr}, \mathsf{bar}, \mathsf{rr}, \mathsf{rrr}, \mathsf{smr}\}\ , \\
B'' &= \mathsf{B_{thr}} \cup \mathsf{MA_{thr}} \cup \{-1\} \cup A \cup \mathbb{B}\ , \\
\mathcal{S}'' &= \{S'' : M'' \to B'' \mid S'' \restriction \mathsf{M_{thr}} \in \mathsf{S_{thr}} \wedge S''(\mathsf{tlr}) \in \mathsf{MA_{thr}} \cup \{-1\} \wedge \\
&\qquad S''(\mathsf{bar}) \in A \wedge S''(\mathsf{rr}) \in \mathbb{B} \wedge S''(\mathsf{rrr}) \in \mathbb{B} \wedge S''(\mathsf{smr}) \in \mathbb{B}\}\ , \\
\mathcal{O}'' &= \{O_{\mathsf{fetch}}\}\ , \\
A'' &= \{\mathsf{fetch}\}\ , \\
[\![\mathsf{fetch}]\!]'' &= (O_{\mathsf{fetch}}, \mathsf{rrr})\ .
\end{aligned}
$$

Here, O_{fetch} is the unique function from $\mathcal{S}''$ to $\mathcal{S}''$ such that for all $S'' \in \mathcal{S}''$:

$$\begin{array}{lll}
O_{\mathsf{fetch}}(S'') \upharpoonright \mathsf{M}_{\mathsf{thr}} = S'' \upharpoonright \mathsf{M}_{\mathsf{thr}}\ , & & \\
O_{\mathsf{fetch}}(S'')(\mathsf{tlr}) & = ntla(S'', r)\ , & \\
O_{\mathsf{fetch}}(S'')(\mathsf{bar}) & = \pi_2(S''(\mathsf{M}_{\mathsf{thr}}[ntla(S'', r)])) & \text{if } S''(\mathsf{M}_{\mathsf{thr}}[ntla(S'', r)]) \notin \{\mathsf{S}, \mathsf{D}\}\ , \\
O_{\mathsf{fetch}}(S'')(\mathsf{bar}) & = S''(\mathsf{bar}) & \text{if } S''(\mathsf{M}_{\mathsf{thr}}[ntla(S'', r)]) \in \{\mathsf{S}, \mathsf{D}\}\ , \\
O_{\mathsf{fetch}}(S'')(\mathsf{rr}) & = S''(\mathsf{rr})\ , & \\
O_{\mathsf{fetch}}(S'')(\mathsf{rrr}) & = \mathsf{T} & \text{if } S''(\mathsf{M}_{\mathsf{thr}}[ntla(S'', r)]) \notin \{\mathsf{S}, \mathsf{D}\}\ , \\
O_{\mathsf{fetch}}(S'')(\mathsf{rrr}) & = \mathsf{F} & \text{if } S''(\mathsf{M}_{\mathsf{thr}}[ntla(S'', r)]) \in \{\mathsf{S}, \mathsf{D}\}\ , \\
O_{\mathsf{fetch}}(S'')(\mathsf{smr}) & = \mathsf{T} & \text{if } S''(\mathsf{M}_{\mathsf{thr}}[ntla(S'', r)]) = \mathsf{S}\ , \\
O_{\mathsf{fetch}}(S'')(\mathsf{smr}) & = \mathsf{F} & \text{if } S''(\mathsf{M}_{\mathsf{thr}}[ntla(S'', r)]) \neq \mathsf{S}\ ,
\end{array}$$

where $r = S''(\mathsf{rr})$, and where $ntla : \mathcal{S}'' \times \mathbb{B} \to \mathsf{MA}_{\mathsf{thr}}$ is defined as in the definition of an ST-4O-enhancement.

To control the execution of a thread, we introduce below control thread CT' for RST-enhancements of Maurer machines and control thread CT'' for remote machines for stored threads. Preceding that, we sketch the behaviour of the cyclic interleaving of CT' and CT''.

While fetch does not fail, fetch and exsba are performed alternatingly. When fetch fails, the cyclic interleaving of CT' and CT'' proceeds as CT'. This means that exsba is performed once more before the whole comes to an end, but that has no effect because rrr contains F.

The guarded recursive specification of CT' consists of the following equation:

$$CT' = CT' \trianglelefteq \mathsf{exsba} \trianglerighteq (\mathsf{S} \trianglelefteq \mathsf{halt} \trianglerighteq \mathsf{D})\ ,$$

and the guarded recursive specification of CT'' consists of the following equation:

$$CT'' = CT'' \trianglelefteq \mathsf{fetch} \trianglerighteq \mathsf{S}\ .$$

Applying thread p to a state of Maurer machine H has the same effect as applying the cyclic interleaving of control threads CT' and CT'', starting with CT'', to the corresponding state of the parallel composition of the RST-enhancement of H and the remote machine for stored threads in which the thread memory contains the stored graph representation of p. This is stated rigorously in the following theorem.

Theorem 11.1. (RST-enhancement)
Let $H' = (M', B', \mathcal{S}', \mathcal{O}', A', [\![\_]\!]')$ be the RST-enhancement of $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$, let H'' be the remote machine for stored threads, let $p \in \mathcal{T}_{\mathsf{finrec}}(A)$ be such that $size(p) \leq size(\mathsf{M}_{\mathsf{thr}})$, let $\mathcal{S}^*$ be the set of states of $H' \parallel H''$, and let $S_0^* \in \mathcal{S}^*$ be such that $S_0^* \upharpoonright \mathsf{M}_{\mathsf{thr}}[0, size(p) - 1] = \mathsf{s}_{\mathsf{thr}}(p)$, $S_0^*(\mathsf{tlr}) = -1$, $S_0^*(\mathsf{rr}) = \mathsf{T}$. Then $p \bullet_H (S_0^* \upharpoonright M) = (\|(\langle CT'' \rangle \curvearrowright \langle CT' \rangle) \bullet_{H' \| H''} S_0^*) \upharpoonright M$.

Proof:
Firstly, $\|(\langle CT'' \rangle \curvearrowright \langle CT' \rangle)$ is the solution of the guarded recursive specification over BTA that consists of the following equation:

$$CT^* = (CT^* \trianglelefteq \mathsf{exsba} \trianglerighteq \|(\langle CT'' \rangle \curvearrowright \langle \mathsf{S} \trianglelefteq \mathsf{halt} \trianglerighteq \mathsf{D} \rangle)) \trianglelefteq \mathsf{fetch} \trianglerighteq CT'\ .$$

Secondly, $H' \parallel H''$ is the Maurer machine $H = (M^*, B^*, \mathcal{S}^*, \mathcal{O}^*, A^*, [\![\_]\!]^*)$ such that

$$\begin{aligned}
M^* &= M \cup \mathsf{M_{thr}} \cup \{\mathsf{tlr}, \mathsf{bar}, \mathsf{rr}, \mathsf{rrr}, \mathsf{smr}\}\ ,\\
B^* &= B \cup \mathsf{B_{thr}} \cup \mathsf{MA_{thr}} \cup \{-1\} \cup A \cup \mathbb{B}\ ,\\
\mathcal{S}^* &= \{S^* : M^* \to B^* \mid S^* \restriction M \in S \land S^* \restriction \mathsf{M_{thr}} \in \mathsf{S_{thr}} \land S^*(\mathsf{tlr}) \in \mathsf{MA_{thr}} \cup \{-1\} \land\\
&\qquad\qquad S^*(\mathsf{bar}) \in A \land S^*(\mathsf{rr}) \in \mathbb{B} \land S^*(\mathsf{rrr}) \in \mathbb{B} \land S^*(\mathsf{smr}) \in \mathbb{B}\}\ ,\\
\mathcal{O}^* &= \{O^* : \mathcal{S}^* \to \mathcal{S}^* \mid \exists O \in \mathcal{O} \bullet \forall S^* \in \mathcal{S}^* \bullet\\
&\qquad (O^*(S^*) \restriction M = O(S^* \restriction M) \land O^*(S^*) \restriction (M^* \setminus M) = S^* \restriction (M^* \setminus M))\}\\
&\quad \cup \{O_{\mathsf{halt}}, O_{\mathsf{fetch}}, O_{\mathsf{exsba}}\}\ ,\\
A^* &= \{\mathsf{halt}, \mathsf{fetch}, \mathsf{exsba}\}\ ,\\
[\![\mathsf{halt}]\!]^* &= (O_{\mathsf{halt}}, \mathsf{rr})\ ,\\
[\![\mathsf{fetch}]\!]^* &= (O_{\mathsf{fetch}}, \mathsf{rrr})\ ,\\
[\![\mathsf{exsba}]\!]^* &= (O_{\mathsf{exsba}}, \mathsf{rrr})\ .
\end{aligned}$$

Here, O_{halt}, O_{fetch} and O_{exsba} are the extensions of the operations O_{halt}, O_{fetch} and O_{exsba} of H' and H'' to S^* such that $O_{\mathsf{halt}}(S^*) \restriction (M^* \setminus M') = S^* \restriction (M^* \setminus M')$, $O_{\mathsf{fetch}}(S^*) \restriction (M^* \setminus M'') = S^* \restriction (M^* \setminus M'')$ and $O_{\mathsf{exsba}}(S^*) \restriction (M^* \setminus M') = S^* \restriction (M^* \setminus M')$.

The remainder of the proof follows the same line as the proof of Theorem 8.1. □

Variations of the way to deal with arbitrary finite-state threads presented above are possible. For example, the fetch and execute operations could have been kept essentially the same as the ones of an ST-4O-enhancement. In that case, test operations would have been needed to check the most recently produced reply of the other Maurer machine. Moreover, a cyclic interleaving strategy would have been needed that gives each control thread two consecutive turns.

12. Stored Threads and Programs

In this section, we discuss the connection between stored threads and programs. First, we review the program notation PGLD, which is close to existing assembly languages. PGLD belongs to a hierarchy of program notations rooted in program algebra. Both program algebra and that hierarchy of program notations are introduced in [6].

In PGLD, it is assumed that there is a fixed but arbitrary set of *basic instructions* $\mathfrak{I}$. PGLD has the following primitive instructions:

- for each $a \in \mathfrak{I}$, a *positive test instruction* $+a$;
- for each $a \in \mathfrak{I}$, a *negative test instruction* $-a$;
- for each $a \in \mathfrak{I}$, a *void basic instruction* a;
- for each $k \in \mathbb{N}$, an *absolute jump instruction* $\#\#k$.

PGLD programs have the form $u_1; \ldots; u_n$ where $u_1, \ldots, u_n$ are primitive instructions of PGLD.

The intuition is that the execution of a basic instruction a may modify a state and produces a Boolean value on completion. In the case of a positive test instruction $+a$, basic instruction a is executed and

Table 12. Defining equations for behaviour extraction

$\vert i, u_1 \,;\, \ldots \,;\, u_n \vert = \mathsf{S}$	if not $1 \leq i \leq n$
$\vert i, u_1 \,;\, \ldots \,;\, u_n \vert = a \circ \vert i+1, u_1 \,;\, \ldots \,;\, u_n \vert$	if $u_i = a$
$\vert i, u_1 \,;\, \ldots \,;\, u_n \vert = \vert i+1, u_1 \,;\, \ldots \,;\, u_n \vert \trianglelefteq a \trianglerighteq \vert i+2, u_1 \,;\, \ldots \,;\, u_n \vert$	if $u_i = +a$
$\vert i, u_1 \,;\, \ldots \,;\, u_n \vert = \vert i+2, u_1 \,;\, \ldots \,;\, u_n \vert \trianglelefteq a \trianglerighteq \vert i+1, u_1 \,;\, \ldots \,;\, u_n \vert$	if $u_i = -a$
$\vert i, u_1 \,;\, \ldots \,;\, u_n \vert = \vert k, u_1 \,;\, \ldots \,;\, u_n \vert$	if $u_i = \#\#k$

execution proceeds with the next primitive instruction if T is produced and otherwise the next primitive instruction is skipped and execution proceeds with the primitive instruction following the skipped one. In the case where T is produced and there is not at least one subsequent primitive instruction and in the case where F is produced and there are not at least two subsequent primitive instructions, termination occurs. In the case of a negative test instruction $-a$, the role of the produced Boolean value is reversed. In the case of a void basic instruction a, the produced Boolean value is disregarded: execution always proceeds with the next primitive instruction (if present). The effect of an absolute jump instruction $\#\#k$ is that execution proceeds with the k-th instruction of the program concerned. If $\#\#k$ is itself the k-th instruction, then inaction (deadlock) occurs. If k equals 0 or k is greater than the length of the program, termination occurs.

We write $\mathcal{P}_{\mathrm{pgld}}$ for the set of all PGLD programs.

The behaviour of a PGLD program is a thread. The function $|\_|_{\mathrm{pgld}}$ that maps each PGLD program to its behaviour is defined by $|u_1 \,;\, \ldots \,;\, u_n|_{\mathrm{pgld}} = |1, u_1 \,;\, \ldots \,;\, u_n|$ where $|\_,\_|$ is defined by the equations given in Table 12. In this table, $u_1, \ldots, u_n$ are primitive instructions of PGLD, $a \in \mathfrak{I}$ and $k, i \in \mathbb{N}$. The equations given in Table 12 do not cover the case where there are cyclic chains of jump instructions. We stipulate that $|i, u_1 \,;\, \ldots \,;\, u_n| = \mathsf{D}$ if u_i is a jump instruction contained in a cyclic chain of jump instructions. It is easy to see that the behaviour of each PGLD program is definable by a finite guarded recursive specification over BTA. Moreover, each finite guarded recursive specification over BTA can be translated to a PGLD program whose behaviour is the solution of the finite guarded recursive specification concerned (cf. Section 5 of [4]).

Next, we consider the stored threads from Section 6 again. We write $\mathcal{ST}$ for $\{\mathsf{s_{thr}}(p) \mid p \in \mathcal{T}_{\mathsf{finrec}} \wedge \mathit{size}(p) \leq \mathit{size}(\mathsf{M_{thr}})\}$. We define a translation function $\mathit{pgld} : \mathcal{ST} \to \mathcal{P}_{\mathrm{pgld}}$ for stored threads. For all $T \in \mathcal{ST}$, $\mathit{pgld}(T) = \mathit{pgld}'(T, 0)$, where $\mathit{pgld}' : \mathcal{ST} \times \mathbb{N} \to \mathcal{P}_{\mathrm{pgld}}$ is recursively defined as follows:

$$\begin{aligned} &\mathit{pgld}'(T,n) = \mathit{pgld}''(T,n) && \text{if } \mathsf{M_{thr}}[n+1] \notin \mathit{dom}(T)\ , \\ &\mathit{pgld}'(T,n) = \mathit{pgld}''(T,n) ; \mathit{pgld}'(T,n+1) && \text{if } \mathsf{M_{thr}}[n+1] \in \mathit{dom}(T)\ , \end{aligned}$$

where $\mathit{pgld}'' : \mathcal{ST} \times \mathbb{N} \to \mathcal{P}_{\mathrm{pgld}}$ is defined as follows:

$$\begin{aligned} &\mathit{pgld}''(T,n) = +a ; \#\#3n'{+}1 ; \#\#3n''{+}1 && \text{if } \mathsf{M_{thr}}[n] = (n', a, n'')\ , \\ &\mathit{pgld}''(T,n) = \#\#0 ; \#\#0 ; \#\#0 && \text{if } \mathsf{M_{thr}}[n] = \mathsf{S}\ , \\ &\mathit{pgld}''(T,n) = \#\#3n{+}1 ; \#\#3n{+}2 ; \#\#3n{+}3 && \text{if } \mathsf{M_{thr}}[n] = \mathsf{D}\ . \end{aligned}$$

The function pgld transforms addresses of thread memory elements containing representations of nodes to absolute jump instructions taking the line that each representation of a node is mapped to three primitive instructions. For that reason, S and D are mapped to three primitive instructions.

It can be shown that, for all $p \in \mathcal{T}_{\mathsf{finrec}}$ with $size(p) \leq size(\mathsf{M}_{\mathsf{thr}})$, $|pgld(\mathsf{s}_{\mathsf{thr}}(p))|_{\mathrm{pgld}} = p$. The function $pgld$ shows that there is hardly a difference between the stored thread $\mathsf{s}_{\mathsf{thr}}(p)$ and the PGLD program $pgld(\mathsf{s}_{\mathsf{thr}}(p))$ extracted from it: $\mathsf{s}_{\mathsf{thr}}(p)$ can also be viewed as a stored representation of $pgld(\mathsf{s}_{\mathsf{thr}}(p))$ with three primitive instruction to a memory element. However, it is likely that $pgld(\mathsf{s}_{\mathsf{thr}}(p))$ contains needless jump instructions. For example, what can be achieved by a positive test instruction $+a$ followed by two identical jump instructions can also be achieved by a void basic instruction a. In other words, PGLD permits a more efficient representation of threads than the one obtained by way of $\mathsf{s}_{\mathsf{thr}}$ and $pgld$.

What is most important for the modelling of micro-architectures is the presence of test and jump instructions in PGLD. The modelling of more advanced micro-architectures must more often than not deal explicitly with test and jump instructions (cf. [8]). This makes stored threads often less adequate when modelling more advanced micro-architectures. In such cases, conversion from stored threads to stored PGLD programs is a feasible option.

An interesting feature of PGLD is that PGLD programs are close to terms of Program Algebra (PGA); and a mapping has been defined by which they can be turned into terms of PGA (see e.g. [6]). Using the axioms of PGA, programs can be simplified algebraically. For example, chained jumps can be removed and thus the size of the program can be reduced.

13. Concluding Remarks

We have investigated basic issues concerning stored threads and their execution on a Maurer machine. We have shown that a single thread can control the execution on a Maurer machine of any executable finite-state thread stored in the memory of the Maurer machine. In fact, that has been done by modelling one of the simplest micro-architectures with single thread control of the execution of stored threads, using Maurer machines and BTA, and verifying that stored threads are executed correctly with the micro-architecture modelled. In a similar manner, we have also shown that finite-state threads of arbitrary size can be dealt with if the Maurer machine on which the execution takes place leaves the fetching of the basic actions to another Maurer machine whose memory size is sufficient for the thread concerned.

The model of one of the simplest micro-architecture with single thread control of the execution of stored threads has been developed gradually via models of semi-micro-architectures. The gradual development clarifies in some degree why virtually all existing micro-architectures for general-purpose computers have grown out of that simple micro-architecture.

We believe that the work presented in this paper demonstrates the feasibility of an approach based on Maurer machines and BTA to model micro-architectures and to verify their correctness and anticipated speed-up results. In [8], we have already made use of the experience gained in this paper to model a micro-architecture with pipelined instruction processing and to verify its correctness. We feel that we were able to model that micro-architecture at the level of abstraction at which micro-architecture design takes place. We are not aware of other approaches where micro-architectures can be modelled at that level of abstraction.

The work presented in this paper, as well as the work presented in [8], was in part carried out in the framework of a project investigating micro-threading [15, 18], a technique for speeding up instruction processing on a computer that makes use of the abilities of the computer to process instructions simultaneously in cases where the state changes involved do not influence each other. This technique requires that programs are parallelized by judicious use of thread forking. After the report version of this paper appeared, we have also investigated parallelization for simple programs, called straight-line programs, using Maurer machines and thread algebra. In that work, which is presented in [10], we focus our attention on basic speed-up results and correctness of program parallelizations.

Acknowledgements

We thank two anonymous referees for carefully reading a preliminary version of this paper and for suggesting improvements of the presentation of the paper.

A. Results on Maurer Computers

In this appendix, we summarize the main results about the composition of operations, the decomposition of operations and the existence of operations with specified input, output and affected regions.

We have the following theorem about the input region and the output region of the composition of two operations.

Theorem A.1. (Composition of operations)
Let $(M, B, \mathcal{S}, \mathcal{O})$ be a Maurer computer, let $O_1, O_2 \in \mathcal{O}$, and let $O' : \mathcal{S} \to \mathcal{S}$ be defined by $O'(S) = O_2(O_1(S))$. Then $IR(O') \subseteq IR(O_1) \cup IR(O_2)$ and $OR(O_1) \setminus OR(O_2) \subseteq OR(O') \subseteq OR(O_1) \cup OR(O_2)$. If $OR(O_1) \cap IR(O_2) = \emptyset$, then $IR(O_2) \subseteq IR(O')$ and $OR(O') = OR(O_1) \cup OR(O_2)$. Moreover, if $OR(O') = OR(O_1) \cup OR(O_2)$ and $OR(O_1) \cap OR(O_2) = \emptyset$, then also $IR(O_1) \subseteq IR(O')$. If $OR(O_1) \cap IR(O_2) = \emptyset$, $IR(O_1) \cap OR(O_2) = \emptyset$ and $OR(O_1) \cap OR(O_2) = \emptyset$, then $O' = O''$ where $O'' : \mathcal{S} \to \mathcal{S}$ is defined by $O''(S) = O_1(O_2(S))$.

We have the following theorem about the decomposition of an operation.

Theorem A.2. (Decomposition of operations)
Let $(M, B, \mathcal{S}, \mathcal{O})$ be a Maurer computer, let $O \in \mathcal{O}$, and let $x \in OR(O) \setminus IR(O)$. Then there exist $O'_1, O'_2 : \mathcal{S} \to \mathcal{S}$ with $O'_2(O'_1(S)) = O(S)$ such that $IR(O'_1) \subseteq IR(O)$, $IR(O'_2) \subseteq IR(O)$, $OR(O'_1) = \{x\}$ and $OR(O'_2) = OR(O) \setminus \{x\}$.

Let $C = (M, B, \mathcal{S}, \mathcal{O})$ be a Maurer computer. Then the *unit component* of C is the set $\{x \in M \mid \exists b \in B \bullet \forall S \in \mathcal{S} \bullet S(x) = b\}$.

We have the following theorem about the existence of operations for arbitrary input and output regions.

Theorem A.3. (Existence of operations (1))
Let $(M, B, \mathcal{S}, \mathcal{O})$ be a Maurer computer, let Z be its unit component, and let $P, Q \subseteq M$. Then there exists a function $O : \mathcal{S} \to \mathcal{S}$ with $IR(O) = P$ and $OR(O) = Q$ iff $P \cap Z = \emptyset$, $Q \cap Z = \emptyset$, and $P \neq \emptyset \Rightarrow Q \neq \emptyset$.

We have the following theorem about the existence of operations for arbitrary input, output and affected regions.

Theorem A.4. (Existence of operations (2))
Let $(M, B, \mathcal{S}, \mathcal{O})$ be a Maurer computer with countable M, let Z be its unit component, let $P, Q \subseteq M$ with $P \cap Z = \emptyset$, $Q \cap Z = \emptyset$, and $P \neq \emptyset \Rightarrow Q \neq \emptyset$, and let $Q_x \subseteq Q$ with $Q_x \neq \emptyset$ for each $x \in P$. Moreover, assume that the following two conditions are satisfied:

- there exist only finitely many $x \in M$ such that $x \in Q_x$, $y \notin Q_y$ for all $y \in M \setminus \{x\}$, and $\mathit{card}(\{b \in B \mid \exists S \in \mathcal{S} \bullet S(x) = b\}) = 2$;
- for all infinite $Q_0 \subseteq \bigcup_{x \in P} Q_x$, the set $\{x \in P \mid Q_x \cap Q_0 \neq \emptyset\}$ is either infinite or contains an element y for which the set $\{b \in B \mid \exists S \in \mathcal{S} \bullet S(y) = b\}$ is infinite.

Then there exists a function $O : \mathcal{S} \to \mathcal{S}$ with $\mathit{IR}(O) = P$, $\mathit{OR}(O) = Q$ and $\mathit{AR}(\{x\}, O) = Q_x$ for each $x \in P$.

Both conditions in Theorem A.4 are satisfied if $\bigcup_{x \in P} Q_x$ is a finite set.

References

[1] Aceto, L., Fokkink, W. J., Verhoef, C.: Structural Operational Semantics, in: *Handbook of Process Algebra* (J. A. Bergstra, A. Ponse, S. A. Smolka, Eds.), Elsevier, Amsterdam, 2001, 197–292.

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

[3] Bergstra, J. A., Bethke, I.: Polarized Process Algebra and Program Equivalence, *Proceedings 30th ICALP* (J. C. M. Baeten, J. K. Lenstra, J. Parrow, G. J. Woeginger, Eds.), LNCS 2719, Springer-Verlag, 2003, 1–21.

[4] Bergstra, J. A., Bethke, I., Ponse, A.: *Decision Problems for Pushdown Threads*, Report PRG0502, Programming Research Group, University of Amsterdam, 2005.

[5] Bergstra, J. A., Klop, J. W.: Process Algebra: Specification and Verification in Bisimulation Semantics, *Proceedings Mathematics and Computer Science II* (M. Hazewinkel, J. K. Lenstra, L. G. L. T. Meertens, Eds.), CWI Monograph 4, North-Holland, 1986, 61–94.

[6] Bergstra, J. A., Loots, M. E.: Program Algebra for Sequential Code, *Journal of Logic and Algebraic Programming*, **51**(2), 2002, 125–156.

[7] Bergstra, J. A., Middelburg, C. A.: A Thread Algebra with Multi-Level Strategic Interleaving, *CiE 2005* (S. B. Cooper, B. Löwe, L. Torenvliet, Eds.), LNCS 3526, Springer-Verlag, 2005, 35–48.

[8] Bergstra, J. A., Middelburg, C. A.: *Maurer Computers for Pipelined Instruction Processing*, Computer Science Report 06-12, Department of Mathematics and Computer Science, Eindhoven University of Technology, March 2006.

[9] Bergstra, J. A., Middelburg, C. A.: Splitting Bisimulations and Retrospective Conditions, *Information and Computation*, **204**(7), 2006, 1083–1138.

[10] Bergstra, J. A., Middelburg, C. A.: *Synchronous Cooperation for Explicit Multi-Threading*, Computer Science Report 06-29, Department of Mathematics and Computer Science, Eindhoven University of Technology, September 2006.

[11] Bergstra, J. A., Middelburg, C. A.: Thread Algebra with Multi-Level Strategies, *Fundamenta Informaticae*, **71**(2/3), 2006, 153–182.

[12] Bergstra, J. A., Middelburg, C. A.: Simulating Turing Machines on Maurer Machines. To appear in *Journal of Applied Logic*, 2007. Preliminary version: Computer Science Report 05-28, Department of Mathematics and Computer Science, Eindhoven University of Technology.

[13] Bergstra, J. A., Middelburg, C. A.: Thread algebra for strategic interleaving. To appear in *Formal Aspects of Computing*, 2007. Preliminary version: Computer Science Report 04-35, Department of Mathematics and Computer Science, Eindhoven University of Technology.

[14] Bergstra, J. A., Ponse, A.: Combining Programs and State Machines, *Journal of Logic and Algebraic Programming*, **51**(2), 2002, 175–192.

[15] Bolychevsky, A., Jesshope, C. R., Muchnick, V.: Dynamic Scheduling in RISC Architectures, *IEE Proceedings Computers and Digital Techniques*, **143**(5), 1996, 309–317.

[16] Hoare, C. A. R.: *Communicating Sequential Processes*, Prentice-Hall, Englewood Cliffs, 1985.

[17] Hopcroft, J. E., Motwani, R., Ullman, J. D.: *Introduction to Automata Theory, Languages and Computation*, Second edition, Addison-Wesley, Reading, MA, 2001.

[18] Jesshope, C. R., Luo, B.: Micro-threading: A New Approach to Future RISC, *Australian Computer Architecture Conference 2000*, IEEE Computer Society Press, 2000, 34–41.

[19] Maurer, W. D.: A Theory of Computer Instructions, *Journal of the ACM*, **13**(2), 1966, 226–235.

[20] Maurer, W. D.: A Theory of Computer Instructions, *Science of Computer Programming*, **60**, 2006, 244–273.

[21] Middelburg, C. A.: An Alternative Formulation of Operational Conservativity with Binding Terms, *Journal of Logic and Algebraic Programming*, **55**(1/2), 2003, 1–19.

[22] Milner, R.: *Communication and Concurrency*, Prentice-Hall, Englewood Cliffs, 1989.