Simulating Turing Machines on Maurer Machines

J. A. Bergstra [a,b], C. A. Middelburg [a,*]

[a] *Programming Research Group, University of Amsterdam, P.O. Box 41882, 1009 DB Amsterdam, Netherlands*

[b] *Department of Philosophy, Utrecht University, P.O. Box 80126, 3508 TC Utrecht, Netherlands*

Abstract

In a previous paper, we used Maurer machines to model and analyse micro-architectures. In the current paper, we investigate the connections between Turing machines and Maurer machines with the purpose to gain an insight into computability issues relating to Maurer machines. We introduce ways to simulate Turing machines on a Maurer machine which, dissenting from the classical theory of computability, take into account that in reality computations always take place on finite machines. In one of those ways, multi-threads and thread forking have an interesting theoretical application.

Key words: Turing machine, Maurer machine, thread algebra, strategic interleaving, thread forking, fair interleaving strategy

1 Introduction

In this paper, we present ways to simulate Turing machines on a Maurer machines which provide insight into the connections between Turing machines, Maurer machines and real computers.

In [21], Maurer proposes a model for computers that is quite different from the well-known models such as register machines, multistack machines and

* Corresponding author.
Email addresses: `J.A.Bergstra@uva.nl` (J. A. Bergstra), `C.A.Middelburg@uva.nl` (C. A. Middelburg).
URLs: `http://www.science.uva.nl/~janb` (J. A. Bergstra), `http://www.science.uva.nl/~kmiddelb` (C. A. Middelburg).

Turing machines (see e.g. [18]). The strength of Maurer's model is that it is close to real computers. Maurer's model is based on the view that a computer has a memory, the contents of all memory elements make up the state of the computer, the computer processes instructions, and the processing of an instruction amounts to performing an operation on the state of the computer which results in changes of the contents of certain memory elements.

In [10], we investigated basic issues concerning stored threads and their execution on a Maurer machine. In that paper, we showed among other things that a single thread can control the execution on a Maurer machine of any finite-state thread stored in the memory of the Maurer machine, provided the basic actions that make up the thread are taken by the Maurer machine as instructions to be processed. To describe threads, we used an extension of basic thread algebra, a form of process algebra introduced in [3] under the name basic polarized process algebra. The purpose of the investigation was to ascertain the feasibility of an approach based on Maurer machines and basic thread algebra to model micro-architectures and to verify their correctness and anticipated speed-up results.

The extension of basic thread algebra used in [10] concerns an operator for interleaving of threads and, for each Maurer machine, an operator for applying a thread to the Maurer machine from a state of the Maurer machine. 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 basic thread algebra to model and analyse micro-architectures? Firstly, 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-architecture must deal. Secondly, general process algebras, such as ACP [1], CCS [23], and CSP [17], are too general for our purpose as well. Basic thread algebra has been designed as an algebra of deterministic sequential processes that interact with a machine. In [7], we show that the processes considered in basic thread algebra can be viewed as processes that are definable over an extension of ACP with conditions introduced in [5]. However, it is quite awkward to describe and analyse processes of this kind using such a general process algebra.

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 and analyse micro-architectures is relatively unknown indeed. In this paper, we investigate the connections between the two models. The purpose of the investigation is to gain an insight into computability issues relating to Maurer machines.

We present in the first place the most obvious way to simulate Turing machines on a Maurer machine. That way illustrates that the test and write operations implicitly performed on steps of a Turing machine must be capable of reading or overwriting the contents of any cell from the infinite number of cells on the tape of the Turing machine – the cell of which the contents is actually read or overwritten depends on the head position. In Maurer's terminology, a test operation has an infinite input region and a write operation has an infinite output region. Real computers do not have such operations. We show that the operations concerned can be replaced by operations with a finite input region and a finite output region if we abandon the restriction to machines with a finite-state control.

In the most obvious way to simulate Turing machines on a Maurer machine, the finite-state control of the Turing machine in question is rendered into a thread, to be applied to the Maurer machine, that is definable by a finite recursive specification. However, the adaptation of this way to simulate Turing machines to the use of operations with a finite input region and a finite output region usually leads to a thread that is only definable by an infinite recursive specification. Thus, one kind of infinity has been replaced by another kind. We show also a way to get round the latter kind of infinity in the case of convergence. The basic ideas are: (a) the thread corresponding to the finite-state control of the Turing machine in question is stored and then executed under control of a thread that makes the head position part of the operations and (b) the controlling thread grows, by means of thread forking, whenever a head position occurs for the first time.

For the last-mentioned way to simulate Turing machines, the operator for interleaving of threads from [10] is adapted to thread forking. The operator is based on the simplest deterministic interleaving strategy, namely cyclic interleaving. In [11], other plausible interleaving strategies are treated. We discuss why the last-mentioned way to simulate Turing machines on a Maurer machine works for other fair interleaving strategies as well.

The structure of this paper is as follows. First of all, we review Maurer's model for computers (Section 2) and basic thread algebra (Section 3). Following this, we extend basic thread algebra with an operator to deal with interleaving of threads (Section 4) and, for each Maurer machine, an operator which allows for threads to transform states of the associated Maurer machine by means of its operations (Section 5). After that, we set out a way to store a finite-state thread in the memory of a Maurer machine for execution on the Maurer machine (Section 6). Next, we review Turing machines (Section 7) and show the most obvious way to simulate Turing machines on Maurer machines (Section 8). Then, we show two ways to simulate Turing machines on Maurer machines by means of operations with a finite input region and a finite output region only (Sections 9 and 10). After that, we discuss why the last way, which uses cyclic interleaving, works for any fair interleaving strategy (Section 11). Finally, we make some concluding remarks (Section 12).

At first sight, Sections 2–5 look to be shortened versions of sections from [10] and Section 6 looks to be copied in full from [10]. We remark that, in Sections 3–6, not all technical details are the same as in [10].

2 Maurer Computers

In this section, we shortly review Maurer computers, i.e. computers as defined by Maurer in [21].

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

- a non-empty set M;
- a set B with $\mathit{card}(B) \geq 2$;
- 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 [21], 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. We return to these conditions, which are met by any real computer, after the introduction of the input region and output region of an operation.

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:

$$\begin{aligned}
&IR(O) = \{x \in M \mid \exists S_1, S_2 \in \mathcal{S} \bullet (\forall z \in M \setminus \{x\} \bullet S_1(z) = S_2(z) \land \\
&\qquad\qquad\qquad\qquad\qquad\qquad \exists y \in OR(O) \bullet O(S_1)(y) \neq O(S_2)(y))\}\ , \\
&OR(O) = \{x \in M \mid \exists S \in \mathcal{S} \bullet S(x) \neq O(S)(x)\}\ .^{1}
\end{aligned}$$

$OR(O)$ 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.

Let $(M, B, \mathcal{S}, \mathcal{O})$ be a Maurer computer, let $S_1, S_2 \in \mathcal{S}$, and let $O \in \mathcal{O}$. Then $S_1 \restriction IR(O) = S_2 \restriction IR(O)$ implies $O(S_1) \restriction OR(O) = O(S_2) \restriction OR(O)$. 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 fundamental property to hold. The first condition on the states of a Maurer computer could be relaxed somewhat (for more details, see [21]).

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 affecting M' under O*, written $RA(M', O)$, and the *region affected by M'' under O*, written $AR(M'', O)$, are the subsets of M defined as follows:

$$\begin{aligned}
&RA(M', O) = \{x \in IR(O) \mid AR(\{x\}, O) \cap M' \neq \emptyset\}\ , \\
&AR(M'', O) = \\
&\quad \{x \in OR(O) \mid \exists S_1, S_2 \in \mathcal{S} \bullet (\forall z \in IR(O) \setminus M'' \bullet S_1(z) = S_2(z) \land \\
&\qquad\qquad\qquad\qquad\qquad\qquad O(S_1)(x) \neq O(S_2)(x))\}\ .
\end{aligned}$$

$AR(M'', O)$ is the set of all elements of $OR(O)$ that are possibly affected by

[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.

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 [21], Maurer gives many results about the relation between the input region and output region of operations, the composition of operations, the decomposition of operations and the existence of operations with specified input, output and affected regions. In [10], we summarize the main results. Recently, a revised and expanded version of [21], which includes all the proofs, has appeared in [22].

3 Basic Thread Algebra

In this section, we review BTA (Basic Thread Algebra), 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* $\mathcal{A}$ with $\mathsf{tau} \notin \mathcal{A}$. We write $\mathcal{A}_{\mathsf{tau}}$ for $\mathcal{A} \cup \{\mathsf{tau}\}$. BTA has the following constants and operators:

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

We use infix notation for postconditional composition. We introduce *action prefixing* as an abbreviation: $a \circ p$, where p is a term of 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. At completion of the processing of the command, the execution environment produces a reply value. This reply is either T or F and is returned to the thread concerned. Let p and q be closed terms 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). The action tau plays a special role. Its execution will never change any state and always produces a positive reply.

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

Table 1
Axiom of BTA

$x \unlhd \mathsf{tau} \unrhd y = x \unlhd \mathsf{tau} \unrhd x$	T1

Table 2
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

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 $t' \unlhd a \unrhd t''$ 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 [2,3]. 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 these 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 the constants of BTA. The constant standing for the unique solution of E for X is denoted by $\langle X|E\rangle$. Moreover, we use the following notation. Let t be a term of BTA and E be a guarded recursive specification. Then we write $\langle t|E\rangle$ for t with, for all $X \in \mathrm{V}(E)$, all occurrences of X in t replaced by $\langle X|E\rangle$. We add the axioms for guarded recursion given in Table 2 to the axioms of BTA. In this table, X, t_X and E stand for an arbitrary variable, an arbitrary term of BTA and an arbitrary guarded recursive specification, respectively. Side conditions are added to restrict the variables, terms and guarded recursive specifications for which X, t_X and E stand. The additional axioms for guarded recursion are known as the recursive definition principle (RDP) and the recursive specification principle (RSP). 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.

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

Table 3
Approximation induction principle

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

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 AIP (Approximation Induction Principle) given in Table 3. Here, following [2,3], 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 member of $\mathcal{A}_{\mathsf{tau}}$. It happens that RSP follows from AIP.

The structural operational semantics of BTA and its extensions with guarded recursion and projection can be found in [11,10].

Henceforth, we write $\mathbb{T}_{\mathsf{finrec}}$ for the set of all terms of BTA with recursion in which no constants $\langle X|E\rangle$ for infinite E occur, and $\mathcal{T}_{\mathsf{finrec}}$ for the set of all closed terms of BTA with recursion in which no constants $\langle X|E\rangle$ for infinite E occur. 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. We write $p \in p'$, where $p, p' \in \mathcal{T}_{\mathsf{finrec}}$, to indicate that p is a subterm of a term $p'' \in \mathcal{T}_{\mathsf{finrec}}$ for which $p' = p''$ is derivable from RDP.

4 Interleaving of Threads and Thread Forking

In this section, we extend BTA with an operator for interleaving of threads that supports thread forking.

It is assumed that the 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.

Several kinds of strategic interleaving have been elaborated in earlier work, see e.g. [11]. In this paper, we only cover one of the simplest interleaving strategies, namely *cyclic interleaving with perfect forking*. Cyclic interleaving

Table 4
Axioms for cyclic interleaving with perfect forking

$\Vert_{\mathsf{f}}(\langle\,\rangle) = \mathsf{S}$	CSIf1
$\Vert_{\mathsf{f}}(\langle \mathsf{S} \rangle \frown \alpha) = \Vert_{\mathsf{f}}(\alpha)$	CSIf2
$\Vert_{\mathsf{f}}(\langle \mathsf{D} \rangle \frown \alpha) = \mathsf{S}_{\mathsf{D}}(\Vert_{\mathsf{f}}(\alpha))$	CSIf3
$\Vert_{\mathsf{f}}(\langle x \trianglelefteq a \trianglerighteq y \rangle \frown \alpha) = \Vert_{\mathsf{f}}(\alpha \frown \langle x \rangle) \trianglelefteq a \trianglerighteq \Vert_{\mathsf{f}}(\alpha \frown \langle y \rangle)$	CSIf4
$\Vert_{\mathsf{f}}(\langle x \trianglelefteq \mathsf{nt}(n) \trianglerighteq y \rangle \frown \alpha) = \mathsf{tau} \circ \Vert_{\mathsf{f}}(\alpha \frown \langle \phi(n) \rangle \frown \langle x \rangle)$	CSIf5

Table 5
Axioms for deadlock at termination

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

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 cyclic permutation. We mean by 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.

It is assumed that a fixed but arbitrary *thread forking* function $\phi : N \to \mathcal{T}_{\mathsf{finrec}}$, where $N \subseteq \mathbb{N}$, has been given. Moreover, it is assumed that $\mathsf{nt}(n) \in \mathcal{A}$ for all $n \in \mathrm{dom}(\phi)$. The basic action $\mathsf{nt}(n)$ represents the act of forking off thread $\phi(n)$. We consider the case where forking off a thread will never be blocked or fail. Therefore, it always produces a positive reply. The action tau arises as the residue of forking off a thread. We write $\mathcal{NT}$ for the set $\{\mathsf{nt}(n) \mid n \in \mathrm{dom}(\phi)\}$. The strategic interleaving operator for cyclic interleaving with perfect forking is denoted by $\Vert_{\mathsf{f}}(\_)$.

The axioms for cyclic interleaving with perfect forking are given in Table 4. [2] In CSIf3, the auxiliary *deadlock at termination* operator $\mathsf{S}_{\mathsf{D}}(\_)$ is used. It turns termination into deadlock. Its axioms are given in Table 5. In Table 4, a stands for an arbitrary member of $\mathcal{A}_{\mathsf{tau}} \setminus \mathcal{NT}$. In Table 5, a stands for an arbitrary member of $\mathcal{A}_{\mathsf{tau}}$.

In [11], we treat several strategies for cyclic interleaving with forking. All of them deal with cases where forking may be blocked and/or may fail. We believe

[2] 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.

that perfect forking is a suitable abstraction when studying the simulation of Turing machines.

The structural operational semantics for cyclic interleaving without forking is given in [11,10]. The adaptation to the case with perfect forking is obvious.

5 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.

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:

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

The members 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 apply operators associated with Maurer machines are related to the apply operators introduced in [12]. They allow for threads to transform states of the associated Maurer machine by means of its operations. Such state transformations produce either a state of the associated Maurer machine or the *undefined state* $\uparrow$. It is assumed that $\uparrow$ is not a state of any Maurer machine. We extend function restriction to $\uparrow$ by stipulating that $\uparrow \restriction M = \uparrow$ for any set M.[3] 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. Moreover, let $(O_a, m_a) = [\![a]\!]$ for all $a \in A$. Then the processing of a basic action a by H amounts to a state change according to the operation O_a. In the resulting state, the reply produced by H is contained in memory element m_a. 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

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

Table 6
Defining equations for apply operator

$x \bullet_H \uparrow = \uparrow$	
$\mathsf{S} \bullet_H S = S$	
$\mathsf{D} \bullet_H S = \uparrow$	
$(\mathsf{tau} \circ x) \bullet_H S = x \bullet_H S$	
$(x \trianglelefteq a \trianglerighteq y) \bullet_H S = x \bullet_H O_a(S)$	if $O_a(S)(m_a) = \mathsf{T}$
$(x \trianglelefteq a \trianglerighteq y) \bullet_H S = y \bullet_H O_a(S)$	if $O_a(S)(m_a) = \mathsf{F}$

Table 7
Rule for divergence

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

all $a \in A$. Then the apply operator $\_ \bullet_H \_$ is defined by the equations given in Table 6 and the rule given in Table 7. 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 7 can be read as follows: if x diverges from S on H, then $x \bullet_H S$ equals $\uparrow$.

We introduce some auxiliary notions, which are useful in proofs.

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

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.

Henceforth, we write $\_ \vdash_H^* \_$ for the reflexive and transitive closure of $\_ \vdash_H \_$.

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 of which the elements can be addressed by means of members of $[0, \mathit{size}(\mathsf{M}_{\mathsf{thr}}) - 1]$. 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 members of $\mathsf{MA}_{\mathsf{thr}}$ 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 $p \in \mathcal{T}_{\mathsf{finrec}}$ be a term not containing tau. Then the *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 $p' \trianglelefteq a \trianglerighteq p'' \in \mathit{Nodes}(p)$, then $p', p'' \in \mathit{Nodes}(p)$;
- if $\langle X_0 | \{X_0 = t_0, \ldots, X_n = t_n\}\rangle \in \mathit{Nodes}(p)$, $\langle t_0 | \{X_0 = t_0, \ldots, X_n = t_n\}\rangle \equiv p' \trianglelefteq a \trianglerighteq p''$, then $p', p'' \in \mathit{Nodes}(p)$.

We write $\mathit{size}(p)$ for $\mathit{card}(\mathit{Nodes}(p))$.

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

Let $p \in \mathcal{T}_{\mathsf{finrec}}$ be a term not containing tau, with $\mathit{size}(p) \leq \mathit{size}(\mathsf{M}_{\mathsf{thr}})$. Then the *stored graph representation* of p, written $\mathsf{s}_{\mathsf{thr}}(p)$, is the unique function $s_{\mathsf{thr}}{:}\mathsf{M}_{\mathsf{thr}}[0, \mathit{size}(p)-1] \to \mathsf{B}_{\mathsf{thr}}[0, \mathit{size}(p)-1]$ such that for all $n \in [0, \mathit{size}(p)-1]$, $s_{\mathsf{thr}}(\mathsf{M}_{\mathsf{thr}}[n]) = \mathit{nrepr}_p(\mathit{node}_p(n))$, where $\mathit{nrepr}_p{:}\mathit{Nodes}(p) \to \mathsf{B}_{\mathsf{thr}}[0, \mathit{size}(p)-1]$ is defined as follows:

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

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

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

In [3], program algebra and a hierarchy of program notations for finite-state threads rooted in program algebra are introduced. Those program notations permit a more efficient stored representation of threads than the one obtained by $\mathsf{s}_{\mathsf{thr}}$ (see also [10]). Moreover, the lower-level program notations, which are close to existing assembly languages, bring with them test and jump instructions. That makes such a program notation useful when investigating issues related to instruction processing, such as pipelining (see also [4]). However, such a program notation would lead to needless complications when investigating ways to simulate Turing machines on Maurer machines. Therefore, we refrain from introducing such a program notation in this paper.

7 Turing Machines

In this section, we define a kind of Turing machines which is at least as powerful as the kinds of Turing machines that are nowadays often considered as standard (cf. [20,18]). That is, each Turing machine of those kinds can be simulated by a Turing machine of the kind defined here. The Turing machines of the kind defined here, called simple Turing machines, are closer to the ones proposed by Turing in [24].

First we give an intuitive description of a simple Turing machine. A simple Turing machine consists of a finite-state *control*, a one-way infinite *tape* and a *tape head*. The control can be in any of a finite number of states. The tape is divided into a countably infinite number of *cells*. Each cell can hold any one of a finite number of *tape symbols*. One of the tape symbols is the *blank* symbol $\Box$. The tape head is always positioned at one of the tape cells. A simple Turing machine makes *steps* based on its current state and the tape symbol held in the cell at which the tape head is positioned. In one step it changes state and either overwrites the cell at which the tape head is positioned with some tape symbol or moves the tape head left or right one cell (but not both).

We will fix on $\{0, 1, \Box\}$ for the set of tape symbols. We write $\mathsf{B}_{\mathsf{tape}}$ for the set $\{0, 1, \Box\}$. We use the *direction symbols* L and R, and the *halt symbol* H. The symbols L, R and H are no tape symbols. We write D_{hd} for the set $\{\mathsf{L}, \mathsf{R}\}$.

A *simple Turing machine* T consists of the following components:

- a finite set Q;
- a function $\delta : Q \times \mathsf{B}_{\mathsf{tape}} \to Q \times (\mathsf{B}_{\mathsf{tape}} \cup \mathsf{D}_{\mathsf{hd}} \cup \{\mathsf{H}\})$;
- an element $q^0 \in Q$.

The members of Q are called the *states*, δ is called the *transition function*, and q^0 is called the *initial state*.

A *contents* of the tape of a simple Turing machine is a function $\tau{:}\mathbb{N} \to \mathsf{B}_{\mathsf{tape}}$ for which there exists an $n \in \mathbb{N}$ such that for all $m \in \mathbb{N}$ with $m \geq n$, $\tau(m) = \Box$. We write $\mathsf{C}_{\mathsf{tape}}$ for the set of all such functions.

Let $T = (Q, \delta, q^0)$ be a simple Turing machine. Then a *configuration* of T is a triple (q, τ, i), where $q \in Q$, $\tau \in \mathsf{C}_{\mathsf{tape}}$ and $i \in \mathbb{N}$. If $q = q^0$, then (q, τ, i) is an *initial* configuration of T. If $\delta(q, \tau(i)) = (q', \mathsf{H})$ for some $q' \in Q$, then (q, τ, i) is a *terminal* configuration of T. Let (q, τ, i) and (q', τ', i') be configurations of T. Then (q', τ', i') is *next* to (q, τ, i) in T if for some $s' \in \mathsf{B}_{\mathsf{tape}} \cup \mathsf{D}_{\mathsf{hd}}$ such that $s' \neq \mathsf{L}$ if $i = 0$:

$$\delta(q, \tau(i)) = (q', s') \;,$$

for all $j \in \mathbb{N}$:

$$\begin{aligned}
\tau'(j) &= s' && \text{if } i = j \wedge s' \in \mathsf{B}_{\mathsf{tape}} \;, \\
\tau'(j) &= \tau(j) && \text{if } i = j \wedge s' \in \mathsf{D}_{\mathsf{hd}} \;, \\
\tau'(j) &= \tau(j) && \text{if } i \neq j \;,
\end{aligned}$$

and

$$
\begin{aligned}
&i' = i && \text{if } s' \in \mathsf{B}_{\mathsf{tape}}\ , \\
&i' = i - 1 && \text{if } s' = \mathsf{L}\ , \\
&i' = i + 1 && \text{if } s' = \mathsf{R}\ .
\end{aligned}
$$

Let $T = (Q, \delta, q^0)$ be a simple Turing machine. Then the *step* relation $\_ \vdash_T \_ \subseteq (Q \times \mathsf{C}_{\mathsf{tape}} \times \mathbb{N}) \times (Q \times \mathsf{C}_{\mathsf{tape}} \times \mathbb{N})$ is defined by $(q, \tau, i) \vdash_T (q', \tau', i')$ iff (q', τ', i') is next to (q, τ, i) in T. A *computation* of T is a finite path $\langle (q_0, \tau_0, i_0), \ldots, (q_n, \tau_n, i_n) \rangle$ in $\_ \vdash_T \_$ such that (q_0, τ_0, i_0) is an initial configuration of T and (q_n, τ_n, i_n) is a terminal configuration of T.[4]

Example 1 *Consider the simple Turing machine* (Q, δ, q^0)*, where:*

$$
\begin{aligned}
Q &= \{q_0, q_1\}\ , \\
\delta(q_0, 0) &= (q_1, 1)\ , \\
\delta(q_0, 1) &= (q_1, 0)\ , \\
\delta(q_0, \Box) &= (q_0, \mathsf{H})\ , \\
\delta(q_1, 0) &= (q_0, \mathsf{R})\ , \\
\delta(q_1, 1) &= (q_0, \mathsf{R})\ , \\
\delta(q_1, \Box) &= (q_0, \mathsf{R})\ , \\
q^0 &= q_0\ .
\end{aligned}
$$

This is a simple Turing machine that, starting from the initial head position, overwrites cells that hold 0 *with* 1 *and cells that hold* 1 *with* 0 *and halts when the first cell holding* $\Box$ *is reached.*

In the case of a simple Turing machine, the set of tape symbols is invariably $\mathsf{B}_{\mathsf{tape}}$, the tape is a one-way infinite tape, in each step either a tape cell is overwritten or the tape head is moved (but not both), input symbols are not distinguished and accepting states are not distinguished (for the roles of input symbols and accepting states, see e.g. [18]). The definitions given above can easily be adapted to the cases where the set of tape symbols is an arbitrary finite set, the tape is a two-way infinite tape, in each step made both a cell is overwritten and the tape head is moved, input symbols are distinguished and/or accepting states are distinguished. However, it happens that each Turing machine of the kinds resulting from such adaptations can

[4] We interpret the usual remark "no left move is permitted when the read-write head is at the [left] boundary" (see e.g. [20]) as "when the read-write head is at the left boundary, a left move impedes making a step but does not give rise to halting".

be simulated by a simple Turing machine (for details, see e.g. [16,14,20,18]). Hence, if each simple Turing machine can be simulated on a Maurer machine, then each Turing machine of those other kinds can be simulated on a Maurer machine as well.

We say that a simple Turing machine T can be simulated on a Maurer machine $H = (M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$ if there exists a thread $p \in \mathcal{T}_{\mathsf{finrec}}(A)$ such that for all computations c of T, there exists a state $S \in \mathcal{S}$ such that the computation of (p, S) on H simulates the computation c. Here, simulation of computations is meant in the sense of automata theory [20,18].

8 Simulation of Turing Machines

In this section, we show the most obvious way to simulate simple Turing machines on a Maurer machine. Henceforth, simple Turing machines will shortly be called Turing machines.

Assume that a fixed but arbitrary countably infinite set $\mathsf{M}_{\mathsf{tape}}$ and a fixed but arbitrary bijection $\mathsf{m}_{\mathsf{tape}} : \mathbb{N} \to \mathsf{M}_{\mathsf{tape}}$ have been given. $\mathsf{M}_{\mathsf{tape}}$ is called a *tape memory.* Let $n \in \mathbb{N}$. Then we write $\mathsf{M}_{\mathsf{tape}}[n]$ for $\mathsf{m}_{\mathsf{tape}}(n)$.

The tape memory is an infinite memory of which the elements can be addressed by means of members of $\mathbb{N}$. The elements of the tape memory contain 0, 1 or $\square$. We write $\mathsf{S}_{\mathsf{tape}}$ for the set of all functions $S_{\mathsf{tape}} : \mathsf{M}_{\mathsf{tape}} \to \mathsf{B}_{\mathsf{tape}}$ for which there exists an $n \in \mathbb{N}$ such that for all $m \in \mathbb{N}$ with $m \geq n$, $S_{\mathsf{tape}}(\mathsf{M}_{\mathsf{tape}}[m]) = \square$.

The Maurer machines M_T, M'_T and M''_T defined in this paper would not be Maurer machines if $\mathsf{S}_{\mathsf{tape}}$ was simply the set of all functions $S_{\mathsf{tape}} : \mathsf{M}_{\mathsf{tape}} \to \mathsf{B}_{\mathsf{tape}}$, for Maurer machines may not have states that differ in the contents of infinitely many memory elements.

The memory of the Maurer machine M_T used to simulate Turing machines consists of a *tape memory* ($\mathsf{M}_{\mathsf{tape}}$), a *head position register* (head) and a *reply register* (rr). Its operation set consists of two *test* operations ($O_{\mathsf{test:0}}$, $O_{\mathsf{test:1}}$), two *write* operations ($O_{\mathsf{write:0}}$, $O_{\mathsf{write:1}}$) and two *move head* operations (O_{movel}, O_{mover}). The basic actions of M_T are $\mathsf{test{:}0}$, $\mathsf{test{:}1}$, $\mathsf{write{:}0}$, $\mathsf{write{:}1}$, movel and mover. They are associated with the operations $O_{\mathsf{test:0}}$, $O_{\mathsf{test:1}}$, $O_{\mathsf{write:0}}$, $O_{\mathsf{write:1}}$, O_{movel} and O_{mover}, respectively.

The tape memory $\mathsf{M}_{\mathsf{tape}}$ corresponds to the tape of a Turing machine. The head position register head is meant for containing the address of the tape memory element that corresponds to the tape cell at which the tape head is positioned. The reply register rr is the memory element in which the reply produced by the operations of M_T is stored. The test operations $O_{\mathsf{test:0}}$ and $O_{\mathsf{test:1}}$ are meant for determining which tape symbol is held in the tape memory element of which the address is contained in head, the write operations $O_{\mathsf{write:0}}$ and $O_{\mathsf{write:1}}$ are meant for overwriting the tape memory element of which the address is contained in head with some tape symbol, and the move operations O_{movel} and O_{mover} are meant for decrementing and incrementing, respectively, the address contained in head by one.

It is assumed that $\mathsf{test}{:}s, \mathsf{write}{:}s \in \mathcal{A}$, for all $s \in \mathsf{B_{tape}}$, and $\mathsf{mover}, \mathsf{movel} \in \mathcal{A}$.

M_T is the Maurer machine $(M, B, \mathcal{S}, \mathcal{O}, A, [\![\_]\!])$ such that

$$\begin{aligned}
M &= \mathsf{M_{tape}} \cup \{\mathsf{head}, \mathsf{rr}\} \;, \\
B &= \mathsf{B_{tape}} \cup \mathbb{N} \cup \mathbb{B} \;, \\
\mathcal{S} &= \{S : M \to B \mid S \restriction \mathsf{M_{tape}} \in \mathsf{S_{tape}} \wedge S(\mathsf{head}) \in \mathbb{N} \wedge S(\mathsf{rr}) \in \mathbb{B}\} \;, \\
\mathcal{O} &= \{O_{\mathsf{test}:s}, O_{\mathsf{write}:s} \mid s \in \mathsf{B_{tape}}\} \cup \{O_{\mathsf{mover}}, O_{\mathsf{movel}}\} \;, \\
A &= \{\mathsf{test}{:}s, \mathsf{write}{:}s \mid s \in \mathsf{B_{tape}}\} \cup \{\mathsf{mover}, \mathsf{movel}\} \;, \\
[\![a]\!] &= (O_a, \mathsf{rr}) \quad \text{for all } a \in A \;.
\end{aligned}$$

For each $s \in \mathsf{B_{tape}}$, $O_{\mathsf{test}:s}$ is the unique function from $\mathcal{S}$ to $\mathcal{S}$ such that for all $S \in \mathcal{S}$:

$$\begin{aligned}
&O_{\mathsf{test}:s}(S) \restriction \mathsf{M_{tape}} = S \restriction \mathsf{M_{tape}} \;, \\
&O_{\mathsf{test}:s}(S)(\mathsf{head}) = S(\mathsf{head}) \;, \\
&O_{\mathsf{test}:s}(S)(\mathsf{rr}) = \mathsf{T} \qquad \text{if } S(\mathsf{M_{tape}}[S(\mathsf{head})]) = s \;, \\
&O_{\mathsf{test}:s}(S)(\mathsf{rr}) = \mathsf{F} \qquad \text{if } S(\mathsf{M_{tape}}[S(\mathsf{head})]) \neq s \;;
\end{aligned}$$

for each $s \in \mathsf{B_{tape}}$, $O_{\mathsf{write}:s}$ is the unique function from $\mathcal{S}$ to $\mathcal{S}$ such that for all $S \in \mathcal{S}$ and $n \in \mathbb{N}$:

$$\begin{aligned}
&O_{\mathsf{write}:s}(S)(\mathsf{M_{tape}}[S(\mathsf{head})]) = s \;, \\
&O_{\mathsf{write}:s}(S)(\mathsf{M_{tape}}[n]) = S(\mathsf{M_{tape}}[n]) \quad \text{if } S(\mathsf{head}) \neq n \;, \\
&O_{\mathsf{write}:s}(S)(\mathsf{head}) = S(\mathsf{head}) \;, \\
&O_{\mathsf{write}:s}(S)(\mathsf{rr}) = \mathsf{T} \;;
\end{aligned}$$

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

$$\begin{aligned}
&O_{\mathsf{mover}}(S) \restriction \mathsf{M_{tape}} = S \restriction \mathsf{M_{tape}} \;, \\
&O_{\mathsf{mover}}(S)(\mathsf{head}) = S(\mathsf{head}) + 1 \;, \\
&O_{\mathsf{mover}}(S)(\mathsf{rr}) = \mathsf{T} \;;
\end{aligned}$$

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

$$\begin{array}{lll}
O_{\mathsf{movel}}(S) \restriction \mathsf{M}_{\mathsf{tape}} & = S \restriction \mathsf{M}_{\mathsf{tape}} \ , & \\
O_{\mathsf{movel}}(S)(\mathsf{head}) & = S(\mathsf{head}) - 1 & \text{if } S(\mathsf{head}) > 0 \ , \\
O_{\mathsf{movel}}(S)(\mathsf{head}) & = 0 & \text{if } S(\mathsf{head}) = 0 \ , \\
O_{\mathsf{movel}}(S)(\mathsf{rr}) & = \mathsf{T} & \text{if } S(\mathsf{head}) > 0 \ , \\
O_{\mathsf{movel}}(S)(\mathsf{rr}) & = \mathsf{F} & \text{if } S(\mathsf{head}) = 0 \ .
\end{array}$$

We write $\mathcal{S}_{M_T}$ and A_{M_T} for the set of states of M_T and the set of basic actions of M_T, respectively.

A *Turing thread* is a constant $\langle X_0 | \{X_0 = t_0, \ldots, X_n = t_n\} \rangle \in \mathcal{T}_{\mathsf{finrec}}$, where $t_0, \ldots, t_n$ are terms of the form $t \trianglelefteq \mathsf{test{:}0} \trianglerighteq (t' \trianglelefteq \mathsf{test{:}1} \trianglerighteq t'')$ with t, t' and t'' of the form $\mathsf{write}{:}s \circ X$ or $\mathsf{mover} \circ X$ or $X \trianglelefteq \mathsf{movel} \trianglerighteq \mathsf{D}$ or S ($s \in \mathsf{B}_{\mathsf{tape}}$, $X \in \{X_0, \ldots, X_n\}$).

A Turing thread corresponds to the finite-state control of a Turing machine. It can be obtained from the transition function of the Turing machine in question in the simple way described at the beginning of the proof of Theorem 2 below. We have $X \trianglelefteq \mathsf{movel} \trianglerighteq \mathsf{D}$ instead of $\mathsf{movel} \circ X$ to deal with the exceptional case where $\mathsf{head} = 0$: $X \trianglelefteq \mathsf{movel} \trianglerighteq \mathsf{D}$ corresponds to "when the tape head is at the left boundary, a left move impedes making a step but does not give rise to halting". Each Turing machine can be simulated on the Maurer machine M_T by means of a Turing thread $p \in \mathcal{T}_{\mathsf{finrec}}(A_{M_T})$. This is stated rigorously in the following theorem.

Theorem 2 *Let $T = (Q, \delta, q^0)$ be a Turing machine. Then there exists a Turing thread $p \in \mathcal{T}_{\mathsf{finrec}}(A_{M_T})$ such that for all computations c of T, there exists an $S \in \mathcal{S}_{M_T}$ such that the computation of (p, S) on M_T simulates c.*

PROOF. Suppose that $Q = \{q_0, \ldots, q_n\}$. Let E be the guarded recursive specification $\{X_i = t_{i0} \trianglelefteq \mathsf{test{:}0} \trianglerighteq (t_{i1} \trianglelefteq \mathsf{test{:}1} \trianglerighteq t_{i\Box}) \mid 0 \leq i \leq n\}$, where

$$\begin{array}{ll}
t_{is} = \mathsf{write}{:}s' \circ X_j & \text{if } \delta(q_i, s) = (q_j, s') \wedge s' \in \mathsf{B}_{\mathsf{tape}} \ , \\
t_{is} = \mathsf{mover} \circ X_j & \text{if } \delta(q_i, s) = (q_j, \mathsf{R}) \ , \\
t_{is} = X_j \trianglelefteq \mathsf{movel} \trianglerighteq \mathsf{D} & \text{if } \delta(q_i, s) = (q_j, \mathsf{L}) \ , \\
t_{is} = \mathsf{S} & \text{if } \exists q \in Q \bullet \delta(q_i, s) = (q, \mathsf{H}) \ .
\end{array}$$

Define $\phi : Q \to \mathcal{T}_{\mathsf{finrec}}(A_{M_T})$ by $\phi(q_i) = \langle X_i | E \rangle$ ($0 \leq i \leq n$). Clearly, $\phi(q_i)$ is a Turing thread. Define $\phi' : \mathsf{C}_{\mathsf{tape}} \times \mathbb{N} \to S_{M_T}$ by $\phi'(\tau, i)$ is the unique state $S \in S_{M_T}$ such that $S(\mathsf{M}_{\mathsf{tape}}[j]) = \tau(j)$ for all $j \in \mathbb{N}$, $S(\mathsf{head}) = i$ and

$S(\mathsf{rr}) = \mathsf{T}$. Combine ϕ and ϕ' to $\phi^* : Q \times \mathsf{C}_{\mathsf{tape}} \times \mathbb{N} \to \mathcal{T}_{\mathsf{finrec}}(A_{M_T}) \times S_{M_T}$ defined by $\phi^*(q, \tau, i) = (\phi(q), \phi'(\tau, i))$. Then we have $(q, \tau, i) \vdash_T (q', \tau', i')$ iff $\phi^*(q, \tau, i) \vdash_{M_T} \phi^*(q', \tau', i')$. This is easily proved by distinction between the following cases: $\delta(q, \tau(i)) \in Q \times \mathsf{B}_{\mathsf{tape}}$, $\delta(q, \tau(i)) \in Q \times \{\mathsf{R}\}$, $\delta(q, \tau(i)) \in Q \times \{\mathsf{L}\}$, $\delta(q, \tau(i)) \in Q \times \{\mathsf{H}\}$. It follows immediately that, if $c = \langle (q_0, \tau_0, i_0), \ldots, (q_n, \tau_n, i_n) \rangle$ is a computation of T, the computation of $(\phi(q_0), \phi'(\tau_0, i_0))$ on M_T simulates c. $\Box$

Example 3 *Consider the Turing machine from Example 1. The Turing thread that corresponds to the finite-state control of this Turing machine is the constant $\langle X_0 | E \rangle$, where*

$$
\begin{aligned}
&E = \\
&\quad \{X_0 = (\mathsf{write{:}1} \circ X_1) \trianglelefteq \mathsf{test{:}0} \trianglerighteq ((\mathsf{write{:}0} \circ X_1) \trianglelefteq \mathsf{test{:}1} \trianglerighteq \mathsf{S}), \\
&\quad\; X_1 = (\mathsf{mover} \circ X_0) \trianglelefteq \mathsf{test{:}0} \trianglerighteq ((\mathsf{mover} \circ X_0) \trianglelefteq \mathsf{test{:}1} \trianglerighteq (\mathsf{mover} \circ X_0))\}\ .
\end{aligned}
$$

The guarded recursive specification E is obtained from the transition function of the Turing machine from Example 1 in the way described at the beginning of the proof of Theorem 2.

Looking at the operations used in the simulation of Turing machines on the Maurer machine M_T, we observe that the test operations $O_{\mathsf{test}:s}$ have an infinite input region and a finite output region and that the write operations $O_{\mathsf{write}:s}$ have a finite input region and an infinite output region. It is easy to see that these infinite regions are essential for many Turing machines. For example, consider the Turing machine from Example 1. This Turing machine overwrites cells that hold 0 with 1 and cells that hold 1 with 0 and halts when the first cell holding $\Box$ is reached. Infinite input and output regions are essential here because the first cell holding $\Box$ may occur anywhere on the tape and Turing machines have only a finite-state control. However, if we expand Turing threads to threads definable by infinite recursive specifications, we can simulate all Turing machines using test and write operations with a finite input region and a finite output region.

9 Using Operations with Finite Input & Output Regions

In order to simulate Turing machines using test and write operations with a finite input region and a finite output region, we have to adapt the Maurer machine M_T. Moreover, for each Turing machine, we have to adapt the corresponding Turing thread. It happens that the Turing threads can be adapted in a uniform way.

We adapt the Maurer machine M_T by removing the head position register head from the memory and the move head operations O_{movel} and O_{mover} from the operation set. In addition, we replace each test operation $O_{\mathsf{test}:s}$ by a test operation $O_{\mathsf{test}:s:n}$ for each $n \in \mathbb{N}$, and each write operation $O_{\mathsf{write}:s}$ by a write operation $O_{\mathsf{write}:s:n}$ for each $n \in \mathbb{N}$. We replace also the basic actions of M_T by basic actions $\mathsf{test}{:}s{:}n$ and $\mathsf{write}{:}s{:}n$ for each $s \in \mathsf{B}_{\mathsf{tape}}$ and $n \in \mathbb{N}$. They are associated with the operations $O_{\mathsf{test}:s:n}$ and $O_{\mathsf{write}:s:n}$, respectively.

The adapted test operations $O_{\mathsf{test}:0:n}$ and $O_{\mathsf{test}:1:n}$ are meant for determining which tape symbol is held in the tape memory element of which the address is n. The adapted write operations $O_{\mathsf{write}:0:n}$ and $O_{\mathsf{write}:1:n}$ are meant for overwriting the tape memory element of which the address is n with some tape symbol.

It is assumed that $\mathsf{test}{:}s{:}n, \mathsf{write}{:}s{:}n \in \mathcal{A}$ for all $s \in \mathsf{B}_{\mathsf{tape}}$ and $n \in \mathbb{N}$.

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

$$\begin{aligned}
M &= \mathsf{M}_{\mathsf{tape}} \cup \{\mathsf{rr}\} \;, \\
B &= \mathsf{B}_{\mathsf{tape}} \cup \mathbb{B} \;, \\
\mathcal{S} &= \{S : M \to B \mid S \restriction \mathsf{M}_{\mathsf{tape}} \in \mathsf{S}_{\mathsf{tape}} \land S(\mathsf{rr}) \in \mathbb{B}\} \;, \\
\mathcal{O} &= \{O_{\mathsf{test}:s:n}, O_{\mathsf{write}:s:n} \mid s \in \mathsf{B}_{\mathsf{tape}} \land n \in \mathbb{N}\} \;, \\
A &= \{\mathsf{test}{:}s{:}n, \mathsf{write}{:}s{:}n \mid s \in \mathsf{B}_{\mathsf{tape}} \land n \in \mathbb{N}\} \;, \\
[\![a]\!] &= (O_a, \mathsf{rr}) \quad \text{for all } a \in A \;.
\end{aligned}$$

For each $s \in \mathsf{B}_{\mathsf{tape}}$ and $n \in \mathbb{N}$, $O_{\mathsf{test}:s:n}$ is the unique function from $\mathcal{S}$ to $\mathcal{S}$ such that for all $S \in \mathcal{S}$:

$$\begin{aligned}
&O_{\mathsf{test}:s:n}(S) \restriction \mathsf{M}_{\mathsf{tape}} = S \restriction \mathsf{M}_{\mathsf{tape}} \;, \\
&O_{\mathsf{test}:s:n}(S)(\mathsf{rr}) = \mathsf{T} && \mathsf{if}\ S(\mathsf{M}_{\mathsf{tape}}[n]) = s \;, \\
&O_{\mathsf{test}:s:n}(S)(\mathsf{rr}) = \mathsf{F} && \mathsf{if}\ S(\mathsf{M}_{\mathsf{tape}}[n]) \neq s \;;
\end{aligned}$$

for each $s \in \mathsf{B}_{\mathsf{tape}}$ and $n \in \mathbb{N}$, $O_{\mathsf{write}:s:n}$ is the unique function from $\mathcal{S}$ to $\mathcal{S}$ such that for all $S \in \mathcal{S}$ and $m \in \mathbb{N}$:

$$\begin{aligned}
&O_{\mathsf{write}:s:n}(S)(\mathsf{M}_{\mathsf{tape}}[n]) = s \;, \\
&O_{\mathsf{write}:s:n}(S)(\mathsf{M}_{\mathsf{tape}}[m]) = S(\mathsf{M}_{\mathsf{tape}}[m]) \quad \mathsf{if}\ n \neq m \;, \\
&O_{\mathsf{write}:s:n}(S)(\mathsf{rr}) = \mathsf{T} \;.
\end{aligned}$$

We write $\mathcal{S}_{M'_T}$ and $A_{M'_T}$ for the set of states of M'_T and the set of basic actions of M'_T, respectively.

For each Turing machine, we have to adapt the corresponding Turing thread to the Maurer machine M'_T. Below, we describe the adaptation concerned in detail.

Let $\langle X_0|E\rangle$, where $E = \{X_0 = t_0, \ldots, X_n = t_n\}$, be a Turing thread, let $i \in \{0, \ldots, n\}$, and let $k \in \mathbb{N}$. Moreover, let T_0 be the set of all terms $t \in \mathbb{T}_{\mathsf{finrec}}$ for which $t_0 = t$ is derivable from E, and let T'_0 be the set of all subterms of some term in T_0. Then the unary relation $\mathcal{HP}_{X_i} \subseteq \mathbb{N}$ is defined by

$$\mathcal{HP}_{X_i}(k) \Leftrightarrow \exists t \in T_0 \bullet \mathcal{HP}'_{X_i}(t, k) \ ,$$

where the auxiliary binary relation $\mathcal{HP}'_{X_i} \subseteq T'_0 \times (\mathbb{N} \cup \{-1\})$ is inductively defined as follows:

- if $X_i \in T'_0$, then $\mathcal{HP}'_{X_i}(X_i, 0)$;
- if $\mathcal{HP}'_{X_i}(t, l)$, $t \trianglelefteq \mathsf{test}{:}s \trianglerighteq t' \in T'_0$ and $l \geq 0$, then $\mathcal{HP}'_{X_i}(t \trianglelefteq \mathsf{test}{:}s \trianglerighteq t', l)$;
- if $\mathcal{HP}'_{X_i}(t, l)$, $t' \trianglelefteq \mathsf{test}{:}s \trianglerighteq t \in T'_0$ and $l \geq 0$, then $\mathcal{HP}'_{X_i}(t' \trianglelefteq \mathsf{test}{:}s \trianglerighteq t, l)$;
- if $\mathcal{HP}'_{X_i}(t, l)$, $\mathsf{write}{:}s \circ t \in T'_0$ and $l \geq 0$, then $\mathcal{HP}'_{X_i}(\mathsf{write}{:}s \circ t, l)$;
- if $\mathcal{HP}'_{X_i}(t, l)$, $\mathsf{mover} \circ t \in T'_0$ and $l \geq 0$, then $\mathcal{HP}'_{X_i}(\mathsf{mover} \circ t, l + 1)$;
- if $\mathcal{HP}'_{X_i}(t, l)$, $t \trianglelefteq \mathsf{movel} \trianglerighteq \mathsf{D} \in T'_0$ and $l \geq 0$, then $\mathcal{HP}'_{X_i}(t \trianglelefteq \mathsf{movel} \trianglerighteq \mathsf{D}, l - 1)$.

$\mathcal{HP}_{X_i}(k)$ indicates that, when $\langle X_0|E\rangle$ at some stage proceeds as $\langle X_i|E\rangle$, k is one of the possible head positions. The recursive specification $\psi(E)$ is inductively defined as follows:

- if $\mathcal{HP}_{X_i}(k)$, then $X_{ik} = t_{ik} \in \psi(E)$,
  where t_{ik} is obtained from t_i by applying the following replacement rules:
  · $\mathsf{test}{:}s$ is replaced by $\mathsf{test}{:}s{:}k$;
  · $\mathsf{write}{:}s \circ X_j$ is replaced by $\mathsf{write}{:}s{:}k \circ X_{jk}$;
  · $\mathsf{mover} \circ X_j$ is replaced by X_{jl}, where $l = k + 1$;
  · if $k \neq 0$, then $X_j \trianglelefteq \mathsf{movel} \trianglerighteq \mathsf{D}$ is replaced by X_{jl}, where $l = k - 1$;
  · if $k = 0$, then $X_j \trianglelefteq \mathsf{movel} \trianglerighteq \mathsf{D}$ is replaced by D.

We write $\psi(\langle X_0|E\rangle)$ for $\langle X_{00}|\psi(E)\rangle$.

The variables of a Turing thread p correspond to the states of a Turing machine. If the head position is made part of the operations, a different copy of a state is needed for each different head position that may occur when the Turing machine enters that state. The variables of $\psi(p)$ correspond to those new states. Consequently, applying Turing thread p to Maurer machine M_T from some state of M_T has the same effect as applying $\psi(p)$ to Maurer machine M'_T from the corresponding state of M'_T. This is stated rigorously in the following theorem.

Theorem 4 *Let p be a Turing thread, and let $S_0 \in \mathcal{S}_{M_T}$ and $S'_0 \in \mathcal{S}_{M'_T}$ be such* *that* $S_0 \restriction (\mathsf{M}_{\mathsf{tape}} \cup \{\mathsf{rr}\}) = S'_0$ *and* $S_0(\mathsf{head}) = 0$. *Then* $(p \bullet_{M_T} S_0) \restriction (\mathsf{M}_{\mathsf{tape}} \cup \{\mathsf{rr}\}) = \psi(p) \bullet_{M'_T} S'_0$.

PROOF. It is easy to see that for all $S \in \mathcal{S}_{M_T}$:

$$\begin{aligned}
O_{\mathsf{test}:s}(S) \restriction (\mathsf{M}_{\mathsf{tape}} \cup \{\mathsf{rr}\}) &= O_{\mathsf{test}:s:n}(S \restriction (\mathsf{M}_{\mathsf{tape}} \cup \{\mathsf{rr}\}))\ , \\
O_{\mathsf{write}:s}(S) \restriction (\mathsf{M}_{\mathsf{tape}} \cup \{\mathsf{rr}\}) &= O_{\mathsf{write}:s:n}(S \restriction (\mathsf{M}_{\mathsf{tape}} \cup \{\mathsf{rr}\}))\ , \\
O_{\mathsf{mover}}(S) \restriction \mathsf{M}_{\mathsf{tape}} &= S \restriction \mathsf{M}_{\mathsf{tape}}\ , \\
O_{\mathsf{movel}}(S) \restriction \mathsf{M}_{\mathsf{tape}} &= S \restriction \mathsf{M}_{\mathsf{tape}}\ ,
\end{aligned}$$

where $n = S(\mathsf{head})$.

Let (p_n, S_n) be the n+1-th element in the full path of (p, S_0) on M_T of which the first component equals S, D or $q \trianglelefteq \mathsf{test}{:}0 \trianglerighteq r$ for some $q, r \in \mathcal{T}_{\mathsf{finrec}}$, and let (p'_n, S'_n) be the n+1-th element in the full path of $(\psi(p), (S_0 \restriction (\mathsf{M}_{\mathsf{tape}} \cup \{\mathsf{rr}\})))$ on M'_T of which the first component equals S, D or $q' \trianglelefteq \mathsf{test}{:}0{:}k \trianglerighteq r'$ for some $k \in \mathbb{N}$ and $q', r' \in \mathcal{T}_{\mathsf{finrec}}$. Then, using the above equations, it is straightforward to prove by induction on n that:

- $p_n = q \trianglelefteq \mathsf{test}{:}0 \trianglerighteq r$ and $S_n(\mathsf{head}) = k$ iff $p'_n = q' \trianglelefteq \mathsf{test}{:}0{:}k \trianglerighteq r'$ with q' and r' obtained from q and r by applying the replacement rules given in the definition of ψ above;
- $S_n \restriction (\mathsf{M}_{\mathsf{tape}} \cup \{\mathsf{rr}\}) = S'_n$.

(if $n < \|(p, S_0)\|_{M_T}$ in case p converges from S_0 on M_T). From this, the theorem follows immediately. □

Theorem 4 deals only with the case where the initial head position is 0. This is sufficient to conclude that each Turing machine can be simulated on the Maurer machine M'_T by means of an adapted Turing thread, for each Turing machine can be simulated by a (simple) Turing machine of which the initial head position is restricted to 0.

Example 5 *Consider again the Turing machine from Example 1. The Turing thread that corresponds to the finite-state control of this Turing machine is given in Example 3. Adaptation of this Turing thread to the case where the head position is made part of the basic actions as described above yields the constant* $\langle X_{00} | E' \rangle$, *where:*

$$\begin{aligned}
E' = \{X_{0k} = (\mathsf{write}{:}1{:}k \circ X_{1k}) \trianglelefteq \mathsf{test}{:}0{:}k \trianglerighteq ((\mathsf{write}{:}0{:}k \circ X_{1k}) \trianglelefteq \mathsf{test}{:}1{:}k \trianglerighteq \mathsf{S}) \\
\mid k \in \mathbb{N}\} \\
\cup \{X_{1k} = X_{0l} \trianglelefteq \mathsf{test}{:}0{:}k \trianglerighteq (X_{0l} \trianglelefteq \mathsf{test}{:}1{:}k \trianglerighteq X_{0l}) \mid k, l \in \mathbb{N} \wedge l = k + 1\}\ .
\end{aligned}$$

Clearly, the adapted thread $\langle X_{00}|E'\rangle$ is an infinite-state thread.

We will show in Section 10 that, when simulating Turing machines on a Maurer machine using test and write operations with a finite input region and a finite output region, we can get round infinite-state threads in the case of convergence.

Hitherto, the results concerning the simulation of Turing machines are closely related to well-known facts about Turing machines. In Maurer's terminology, the facts concerned can be phrased as follows:

- the test operations implicitly performed on steps of a Turing machine have an infinite input region and a finite output region;
- the write operations implicitly performed on steps of a Turing machine have a finite input region and an infinite output region;
- these operations can be replaced by operations with a finite input region and a finite output region if we allow Turing machines with an infinite set of states.

10 Using a Multi-thread and Thread Forking

In this section, we show a way to simulate Turing machines on a Maurer machine using test and write operations with a finite input region and a finite output region that gets round infinite-state threads in the case of convergence. The basic ideas behind it are as follows:

- the thread corresponding to the finite-state control of the Turing machine in question is stored and then executed under control of a multi-thread that makes the head position part of the operations;
- this multi-thread forks off for every head position a control thread of itself on the head reaching the preceding position for the first time.

By using thread forking in this way, the execution remains controlled by a finite thread in the case of convergence. Although it is not necessary to get round infinite-state threads, the head position register head will be replaced by a countably infinite memory.

It is assumed that a fixed but arbitrary countably infinite set M_{hd} and a fixed but arbitrary bijection $\mathsf{m}_{\mathsf{hd}} : \mathbb{N} \to \mathsf{M}_{\mathsf{hd}}$ have been given. M_{hd} is called a *head position memory*. Let $n \in \mathbb{N}$. Then we write $\mathsf{M}_{\mathsf{hd}}[n]$ for $\mathsf{m}_{\mathsf{hd}}(n)$. M_{hd} is an infinite memory of which the elements can be addressed by means of members of $\mathbb{N}$. The elements of M_{hd} contain T or F. We write S_{hd} for the set of all functions $S_{\mathsf{hd}} : \mathsf{M}_{\mathsf{hd}} \to \mathbb{B}$ for which there exists an $n \in \mathbb{N}$ such that

$S_{\mathsf{hd}}(\mathsf{M}_{\mathsf{hd}}[n]) = \mathsf{T}$ and for all $m \in \mathbb{N}$ with $m \neq n$, $S_{\mathsf{hd}}(\mathsf{M}_{\mathsf{hd}}[m]) = \mathsf{F}$. M_{hd} will be used in such a way that the head position is always the unique n for which $\mathsf{M}_{\mathsf{hd}}[n]$ contains T. The memory of the simulating Maurer machine remains countably infinite if head is replaced by M_{hd}. The replacement achieves that, for each memory element, all possible contents belong to a finite set.

The Maurer machine M''_T defined below would not be a Maurer machine if S_{hd} was simply the set of all functions $S_{\mathsf{hd}} : \mathsf{M}_{\mathsf{hd}} \to \mathbb{B}$, for Maurer machines may not have states that differ in the contents of infinitely many memory elements.

We adapt the Maurer machine M_T by extending the memory with a *thread memory* ($\mathsf{M}_{\mathsf{thr}}$), a *thread location register* (tlr) and a *basic action register* (bar), and the operation set with a *halt* operation (O_{halt}), a *fetch* operation (O_{fetch}), an *execute stored basic action* operation ($O_{\mathsf{exsba}:n}$) for each $n \in \mathbb{N}$, a *test execution mode* operation (O_{testem}), and a *test head position* operation ($O_{\mathsf{testhp}:n}$) for each $n \in \mathbb{N}$. We replace the basic actions of M_T by basic actions halt, fetch, $\mathsf{exsba}{:}n$ for each $n \in \mathbb{N}$, testem and $\mathsf{testhp}{:}n$ for each $n \in \mathbb{N}$. They are associated with the operations O_{halt}, O_{fetch}, $O_{\mathsf{exsba}:n}$, O_{testem} and $O_{\mathsf{testhp}:n}$, respectively. In addition, we replace the head position register head by a head position memory M_{hd}, each test operation $O_{\mathsf{test}:s}$ by a test operation $O_{\mathsf{test}:s:n}$ for each $n \in \mathbb{N}$, and each write operation $O_{\mathsf{write}:s}$ by a write operation $O_{\mathsf{write}:s:n}$ for each $n \in \mathbb{N}$.

The thread memory $\mathsf{M}_{\mathsf{thr}}$ is meant for storing a Turing thread p. Processing of a basic action performed by p now amounts to first fetching the basic action from $\mathsf{M}_{\mathsf{thr}}$ in the basic action register bar and then executing the basic action in bar. 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 at 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. The operation $O_{\mathsf{exsba}:n}$ allows for making the head position part of the operation that corresponds to the basic action in bar. The operation $O_{\mathsf{testhp}:n}$ is meant for testing whether the head position is n. The operation O_{testem} allows for testing whether the execution of the stored Turing thread has not yet come to an end.

Once again, it is assumed that $\mathsf{test}{:}s, \mathsf{write}{:}s \in \mathcal{A}$, for all $s \in \mathsf{B}_{\mathsf{tape}}$, and $\mathsf{mover}, \mathsf{movel} \in \mathcal{A}$. Moreover, it is assumed that $\mathsf{testem}, \mathsf{halt}, \mathsf{fetch} \in \mathcal{A}$ and $\mathsf{testhp}{:}n$, $\mathsf{exsba}{:}n \in \mathcal{A}$ for all $n \in \mathbb{N}$.

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

$$
\begin{aligned}
M &= \mathsf{M}_{\mathsf{tape}} \cup \mathsf{M}_{\mathsf{hd}} \cup \mathsf{M}_{\mathsf{thr}} \cup \{\mathsf{tlr}, \mathsf{bar}, \mathsf{rr}\}\ , \\
B &= \mathsf{B}_{\mathsf{tape}} \cup \mathbb{B} \cup \mathsf{B}_{\mathsf{thr}} \cup \mathsf{MA}_{\mathsf{thr}} \cup \{-1\} \cup A_{M_T}\ , \\
\mathcal{S} &= \{S : M \to B \mid \\
&\qquad S \restriction \mathsf{M}_{\mathsf{tape}} \in \mathsf{S}_{\mathsf{tape}} \land S \restriction \mathsf{M}_{\mathsf{hd}} \in \mathsf{S}_{\mathsf{hd}} \land S \restriction \mathsf{M}_{\mathsf{thr}} \in \mathsf{S}_{\mathsf{thr}} \land \\
&\qquad S(\mathsf{tlr}) \in \mathsf{MA}_{\mathsf{thr}} \cup \{-1\} \land S(\mathsf{bar}) \in A_{M_T} \land S(\mathsf{rr}) \in \mathbb{B}\}\ , \\
\mathcal{O} &= \{O_{\mathsf{testem}}, O_{\mathsf{halt}}, O_{\mathsf{fetch}}\} \cup \{O_{\mathsf{testhp}:n}, O_{\mathsf{exsba}:n} \mid n \in \mathbb{N}\} \\
&\qquad \cup \{O_{\mathsf{test}:s:n}, O_{\mathsf{write}:s:n} \mid s \in \mathsf{B}_{\mathsf{tape}} \land n \in \mathbb{N}\} \cup \{O_{\mathsf{mover}}, O_{\mathsf{movel}}\}\ , \\
A &= \{\mathsf{testem}, \mathsf{halt}, \mathsf{fetch}\} \cup \{\mathsf{testhp}{:}n, \mathsf{exsba}{:}n \mid n \in \mathbb{N}\}\ , \\
[\![a]\!] &= (O_a, \mathsf{rr}) \quad \text{for all } a \in A\ .
\end{aligned}
$$

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

$$
\begin{aligned}
O_{\mathsf{testem}}(S) \restriction \mathsf{M}_{\mathsf{tape}} &= S \restriction \mathsf{M}_{\mathsf{tape}}\ , \\
O_{\mathsf{testem}}(S) \restriction \mathsf{M}_{\mathsf{hd}} &= S \restriction \mathsf{M}_{\mathsf{hd}}\ , \\
O_{\mathsf{testem}}(S) \restriction \mathsf{M}_{\mathsf{thr}} &= S \restriction \mathsf{M}_{\mathsf{thr}}\ , \\
O_{\mathsf{testem}}(S)(\mathsf{tlr}) &= S(\mathsf{tlr})\ , \\
O_{\mathsf{testem}}(S)(\mathsf{bar}) &= S(\mathsf{bar})\ , \\
O_{\mathsf{testem}}(S)(\mathsf{rr}) &= \mathsf{T} \qquad \text{if } S(\mathsf{M}_{\mathsf{thr}}[S(\mathsf{tlr})]) \in \{\mathsf{S}, \mathsf{D}\}\ , \\
O_{\mathsf{testem}}(S)(\mathsf{rr}) &= \mathsf{F} \qquad \text{if } S(\mathsf{M}_{\mathsf{thr}}[S(\mathsf{tlr})]) \notin \{\mathsf{S}, \mathsf{D}\}\ ;
\end{aligned}
$$

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

$$
\begin{aligned}
O_{\mathsf{halt}}(S) \restriction \mathsf{M}_{\mathsf{tape}} &= S \restriction \mathsf{M}_{\mathsf{tape}}\ , \\
O_{\mathsf{halt}}(S) \restriction \mathsf{M}_{\mathsf{hd}} &= S \restriction \mathsf{M}_{\mathsf{hd}}\ , \\
O_{\mathsf{halt}}(S) \restriction \mathsf{M}_{\mathsf{thr}} &= S \restriction \mathsf{M}_{\mathsf{thr}}\ , \\
O_{\mathsf{halt}}(S)(\mathsf{tlr}) &= S(\mathsf{tlr})\ , \\
O_{\mathsf{halt}}(S)(\mathsf{bar}) &= S(\mathsf{bar})\ , \\
O_{\mathsf{halt}}(S)(\mathsf{rr}) &= \mathsf{T} \qquad \text{if } S(\mathsf{M}_{\mathsf{thr}}[S(\mathsf{tlr})]) = \mathsf{S}\ , \\
O_{\mathsf{halt}}(S)(\mathsf{rr}) &= \mathsf{F} \qquad \text{if } S(\mathsf{M}_{\mathsf{thr}}[S(\mathsf{tlr})]) \neq \mathsf{S}\ ;
\end{aligned}
$$

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

$$\begin{aligned}
&O_{\mathsf{fetch}}(S) \restriction \mathsf{M}_{\mathsf{tape}} = S \restriction \mathsf{M}_{\mathsf{tape}}\ , \\
&O_{\mathsf{fetch}}(S) \restriction \mathsf{M}_{\mathsf{hd}} \quad = S \restriction \mathsf{M}_{\mathsf{hd}}\ , \\
&O_{\mathsf{fetch}}(S) \restriction \mathsf{M}_{\mathsf{thr}} \quad = S \restriction \mathsf{M}_{\mathsf{thr}}\ , \\
&O_{\mathsf{fetch}}(S)(\mathsf{tlr}) \qquad = \mathit{ntl}(S, r)\ , \\
&O_{\mathsf{fetch}}(S)(\mathsf{bar}) \qquad = \pi_2(S(\mathsf{M}_{\mathsf{thr}}[\mathit{ntl}(S,r)])) \quad \text{if } S(\mathsf{M}_{\mathsf{thr}}[\mathit{ntl}(S,r)]) \notin \{\mathsf{S}, \mathsf{D}\}\ , \\
&O_{\mathsf{fetch}}(S)(\mathsf{bar}) \qquad = S(\mathsf{bar}) \qquad\qquad\qquad\quad \text{if } S(\mathsf{M}_{\mathsf{thr}}[\mathit{ntl}(S,r)]) \in \{\mathsf{S}, \mathsf{D}\}\ , \\
&O_{\mathsf{fetch}}(S)(\mathsf{rr}) \qquad\ \ = \mathsf{T} \qquad\qquad\qquad\qquad\quad\ \ \text{if } S(\mathsf{M}_{\mathsf{thr}}[\mathit{ntl}(S,r)]) \notin \{\mathsf{S}, \mathsf{D}\}\ , \\
&O_{\mathsf{fetch}}(S)(\mathsf{rr}) \qquad\ \ = \mathsf{F} \qquad\qquad\qquad\qquad\quad\ \ \text{if } S(\mathsf{M}_{\mathsf{thr}}[\mathit{ntl}(S,r)]) \in \{\mathsf{S}, \mathsf{D}\}\ ,
\end{aligned}$$

where $r = S(\mathsf{rr})$ and $\mathit{ntl} : \mathcal{S} \times \mathbb{B} \to \mathsf{MA}_{\mathsf{thr}}$ is defined as follows:

$$\begin{aligned}
&\mathit{ntl}(S, \mathsf{T}) = \pi_1(S(\mathsf{M}_{\mathsf{thr}}[S(\mathsf{tlr})])) \ \text{if } S(\mathsf{tlr}) \in \mathsf{MA}_{\mathsf{thr}} \land S(\mathsf{M}_{\mathsf{thr}}[S(\mathsf{tlr})]) \notin \{\mathsf{S}, \mathsf{D}\}\,, \\
&\mathit{ntl}(S, \mathsf{F}) = \pi_3(S(\mathsf{M}_{\mathsf{thr}}[S(\mathsf{tlr})])) \ \text{if } S(\mathsf{tlr}) \in \mathsf{MA}_{\mathsf{thr}} \land S(\mathsf{M}_{\mathsf{thr}}[S(\mathsf{tlr})]) \notin \{\mathsf{S}, \mathsf{D}\}\,, \\
&\mathit{ntl}(S, r') = S(\mathsf{tlr}) \qquad\qquad\quad\ \ \text{if } S(\mathsf{tlr}) \in \mathsf{MA}_{\mathsf{thr}} \land S(\mathsf{M}_{\mathsf{thr}}[S(\mathsf{tlr})]) \in \{\mathsf{S}, \mathsf{D}\}\,, \\
&\mathit{ntl}(S, r') = 0 \qquad\qquad\qquad\quad\ \ \text{if } S(\mathsf{tlr}) \notin \mathsf{MA}_{\mathsf{thr}}\ ;
\end{aligned}$$

for each $n \in \mathbb{N}$, $O_{\mathsf{testhp}:n}$ is the unique function from $\mathcal{S}$ to $\mathcal{S}$ such that for all $S \in \mathcal{S}$:

$$\begin{aligned}
&O_{\mathsf{testhp}:n}(S) \restriction \mathsf{M}_{\mathsf{tape}} = S \restriction \mathsf{M}_{\mathsf{tape}}\ , \\
&O_{\mathsf{testhp}:n}(S) \restriction \mathsf{M}_{\mathsf{hd}} \quad = S \restriction \mathsf{M}_{\mathsf{hd}}\ , \\
&O_{\mathsf{testhp}:n}(S) \restriction \mathsf{M}_{\mathsf{thr}} \quad = S \restriction \mathsf{M}_{\mathsf{thr}}\ , \\
&O_{\mathsf{testhp}:n}(S)(\mathsf{tlr}) \qquad = S(\mathsf{tlr})\ , \\
&O_{\mathsf{testhp}:n}(S)(\mathsf{bar}) \qquad = S(\mathsf{bar})\ , \\
&O_{\mathsf{testhp}:n}(S)(\mathsf{rr}) \qquad\ \ = S(\mathsf{M}_{\mathsf{hd}}[n])\ ;
\end{aligned}$$

for each $n \in \mathbb{N}$, $O_{\mathsf{exsba}:n}$ is the unique function from $\mathcal{S}$ to $\mathcal{S}$ such that for all $S \in \mathcal{S}$:

$$O_{\mathsf{exsba}:n}(S) = \mathit{tmi}(S(\mathsf{bar}), n)(S)\ ,$$

where $\mathit{tmi} : A_{M_T} \times \mathbb{N} \to \mathcal{O}$ is defined as follows:

$$\begin{aligned}
&\mathit{tmi}(\mathsf{test}{:}s, n) \ \ = O_{\mathsf{test}:s:n}\ , \\
&\mathit{tmi}(\mathsf{write}{:}s, n) = O_{\mathsf{write}:s:n}\ , \\
&\mathit{tmi}(\mathsf{mover}, n) \ \ = O_{\mathsf{mover}}\ , \\
&\mathit{tmi}(\mathsf{movel}, n) \ \ \ = O_{\mathsf{movel}}\ ;
\end{aligned}$$

for each $s \in \mathsf{B}_{\mathsf{tape}}$ and $n \in \mathbb{N}$, $O_{\mathsf{test}:s:n}$ is the unique function from $\mathcal{S}$ to $\mathcal{S}$ such that for all $S \in \mathcal{S}$:

$$\begin{aligned}
O_{\mathsf{test}:s:n}(S) \restriction \mathsf{M}_{\mathsf{tape}} &= S \restriction \mathsf{M}_{\mathsf{tape}} \;, \\
O_{\mathsf{test}:s:n}(S) \restriction \mathsf{M}_{\mathsf{hd}} &= S \restriction \mathsf{M}_{\mathsf{hd}} \;, \\
O_{\mathsf{test}:s:n}(S) \restriction \mathsf{M}_{\mathsf{thr}} &= S \restriction \mathsf{M}_{\mathsf{thr}} \;, \\
O_{\mathsf{test}:s:n}(S)(\mathsf{tlr}) &= S(\mathsf{tlr}) \;, \\
O_{\mathsf{test}:s:n}(S)(\mathsf{bar}) &= S(\mathsf{bar}) \;, \\
O_{\mathsf{test}:s:n}(S)(\mathsf{rr}) &= \mathsf{T} && \text{if } S(\mathsf{M}_{\mathsf{tape}}[n]) = s \;, \\
O_{\mathsf{test}:s:n}(S)(\mathsf{rr}) &= \mathsf{F} && \text{if } S(\mathsf{M}_{\mathsf{tape}}[n]) \neq s \;;
\end{aligned}$$

for each $s \in \mathsf{B}_{\mathsf{tape}}$ and $n \in \mathbb{N}$, $O_{\mathsf{write}:s:n}$ is the unique function from $\mathcal{S}$ to $\mathcal{S}$ such that for all $S \in \mathcal{S}$ and $m \in \mathbb{N}$:

$$\begin{aligned}
O_{\mathsf{write}:s:n}(S)(\mathsf{M}_{\mathsf{tape}}[n]) &= s \;, \\
O_{\mathsf{write}:s:n}(S)(\mathsf{M}_{\mathsf{tape}}[m]) &= S(\mathsf{M}_{\mathsf{tape}}[m]) && \text{if } n \neq m \;, \\
O_{\mathsf{write}:s:n}(S) \restriction \mathsf{M}_{\mathsf{hd}} &= S \restriction \mathsf{M}_{\mathsf{hd}} \;, \\
O_{\mathsf{write}:s:n}(S) \restriction \mathsf{M}_{\mathsf{thr}} &= S \restriction \mathsf{M}_{\mathsf{thr}} \;, \\
O_{\mathsf{write}:s:n}(S)(\mathsf{tlr}) &= S(\mathsf{tlr}) \;, \\
O_{\mathsf{write}:s:n}(S)(\mathsf{bar}) &= S(\mathsf{bar}) \;, \\
O_{\mathsf{write}:s:n}(S)(\mathsf{rr}) &= \mathsf{T} \;;
\end{aligned}$$

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

$$\begin{aligned}
O_{\mathsf{mover}}(S) \restriction \mathsf{M}_{\mathsf{tape}} &= S \restriction \mathsf{M}_{\mathsf{tape}} \;, \\
O_{\mathsf{mover}}(S) \restriction \mathsf{M}_{\mathsf{hd}} &= \mathit{shiftr}(S \restriction \mathsf{M}_{\mathsf{hd}}) \;, \\
O_{\mathsf{mover}}(S) \restriction \mathsf{M}_{\mathsf{thr}} &= S \restriction \mathsf{M}_{\mathsf{thr}} \;, \\
O_{\mathsf{mover}}(S)(\mathsf{tlr}) &= S(\mathsf{tlr}) \;, \\
O_{\mathsf{mover}}(S)(\mathsf{bar}) &= S(\mathsf{bar}) \;, \\
O_{\mathsf{mover}}(S)(\mathsf{rr}) &= \mathsf{T} \;,
\end{aligned}$$

where $\mathit{shiftr} : \mathsf{S}_{\mathsf{hd}} \to \mathsf{S}_{\mathsf{hd}}$ is defined as follows ($n \in \mathbb{N}$):

$$\begin{aligned}
\mathit{shiftr}(S)(\mathsf{M}_{\mathsf{hd}}[0]) &= \mathsf{F} \;, \\
\mathit{shiftr}(S)(\mathsf{M}_{\mathsf{hd}}[n+1]) &= S(\mathsf{M}_{\mathsf{hd}}[n]) \;;
\end{aligned}$$

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

$$\begin{aligned}
O_{\mathsf{movel}}(S) \restriction \mathsf{M}_{\mathsf{tape}} &= S \restriction \mathsf{M}_{\mathsf{tape}}\ , \\
O_{\mathsf{movel}}(S) \restriction \mathsf{M}_{\mathsf{hd}} &= \mathit{shiftl}(S \restriction \mathsf{M}_{\mathsf{hd}})\ , \\
O_{\mathsf{movel}}(S) \restriction \mathsf{M}_{\mathsf{thr}} &= S \restriction \mathsf{M}_{\mathsf{thr}}\ , \\
O_{\mathsf{movel}}(S)(\mathsf{tlr}) &= S(\mathsf{tlr})\ , \\
O_{\mathsf{movel}}(S)(\mathsf{bar}) &= S(\mathsf{bar})\ , \\
O_{\mathsf{movel}}(S)(\mathsf{rr}) &= \neg\, S(\mathsf{M}_{\mathsf{hd}}[0])\ ,
\end{aligned}$$

where $\mathit{shiftl} : \mathsf{S}_{\mathsf{hd}} \to \mathsf{S}_{\mathsf{hd}}$ is defined as follows ($n \in \mathbb{N}$):

$$\begin{aligned}
\mathit{shiftl}(S)(\mathsf{M}_{\mathsf{hd}}[0]) &= \mathsf{T} && \text{if } S(\mathsf{M}_{\mathsf{hd}}[0]) = \mathsf{T}\ , \\
\mathit{shiftl}(S)(\mathsf{M}_{\mathsf{hd}}[0]) &= S(\mathsf{M}_{\mathsf{hd}}[1]) && \text{if } S(\mathsf{M}_{\mathsf{hd}}[0]) = \mathsf{F}\ , \\
\mathit{shiftl}(S)(\mathsf{M}_{\mathsf{hd}}[n+1]) &= S(\mathsf{M}_{\mathsf{hd}}[n+2])\ .
\end{aligned}$$

To control the execution of a stored Turing thread, we introduce below a control thread CT_n for each head position $n \in \mathbb{N}$. Preceding that, we sketch the behaviour of CT_n, considering that it is subject to cyclic interleaving with other such control threads. Consider a turn of CT_n on which it tests whether the head position is n. If the test succeeds, then CT_n fetches a basic action from the stored Turing thread on its next turn, executes that basic action on its second next turn and tests again whether the head position is n on its third next turn. If the test fails, CT_n tests whether the execution of the stored Turing thread has not yet come to an end on its next turn. If the latter test succeeds, CT_n tests again whether the head position is n on its second next turn. If the latter test fails, CT_n terminates. Their are two exceptions to the behaviour of CT_n sketched above. Firstly, on the first time that the test whether the head position is n succeeds, CT_n forks off the control thread CT_{n+1} between the first successful test and the first fetch. Secondly, in the case where fetching of a basic action fails because there are no more actions to be fetched, CT_n determines on the following turn how the execution of the stored Turing thread should come to an end and acts in accordance with the outcome.

Let $n \in \mathbb{N}$, and let CE_n be the guarded recursive specification over BTA that consists of the following equations:

$$\begin{aligned}
CT_n &= (\mathsf{nt}(n+1) \circ CT'_n) \trianglelefteq \mathsf{testhp}{:}n \trianglerighteq (CT_n \trianglelefteq \mathsf{testem} \trianglerighteq \mathsf{S})\ , \\
CT'_n &= (\mathsf{exsba}{:}n \circ CT''_n) \trianglelefteq \mathsf{fetch} \trianglerighteq (\mathsf{S} \trianglelefteq \mathsf{halt} \trianglerighteq \mathsf{D})\ , \\
CT''_n &= CT'_n \trianglelefteq \mathsf{testhp}{:}n \trianglerighteq (CT''_n \trianglelefteq \mathsf{testem} \trianglerighteq \mathsf{S})\ .
\end{aligned}$$

Moreover, take the function ϕ from $\mathbb{N}$ to $\mathcal{T}_{\mathsf{finrec}}$ defined by $\phi(n) = \langle CT_n | CE_n \rangle$ as the thread forking function. Then applying Turing thread p to the Maurer machine M_T from some state of M_T in which the head position is 0 has the same effect as applying $\|_{\mathrm{f}}(\langle CT_0 \rangle)$ to the Maurer machine M''_T from the corresponding state of M''_T in which the thread memory contains the stored graph representation of p. This is stated rigorously in the following theorem.

Theorem 6 *Let p be a Turing thread such that $size(p) \leq size(\mathsf{M_{thr}})$, and let $S_0 \in \mathcal{S}_{M_T}$ and $S''_0 \in \mathcal{S}_{M''_T}$ be such that $S_0 \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\}) = S''_0 \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\})$, $S_0(\mathsf{head}) = 0$, $S''_0(\mathsf{M_{hd}}[0]) = \mathsf{T}$, $S_0 \restriction \mathsf{M_{thr}}[0, size(p) - 1] = \mathsf{s_{thr}}(p)$, $S_0(\mathsf{tlr}) = -1$. Then $(p \bullet_{M_T} S_0) \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\}) = (\|_{\mathrm{f}}(\langle CT_0 \rangle) \bullet_{M''_T} S''_0) \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\})$ and, for all $n \in \mathbb{N}$, $(p \bullet_{M_T} S_0)(\mathsf{head}) = n$ iff $(\|_{\mathrm{f}}(\langle CT_0 \rangle) \bullet_{M''_T} S''_0)(\mathsf{M_{hd}}[n]) = \mathsf{T}$.*

PROOF. It is easy to see that for all $S \in \mathcal{S}_{M_T}$, $S'' \in \mathcal{S}_{M''_T}$ and $n, n' \in \mathbb{N}$ such that $S \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\}) = S'' \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\})$, $S(\mathsf{head}) = n$ and $S''(\mathsf{M_{hd}}[n]) = \mathsf{T}$:

$$\begin{aligned}
O_{\mathsf{test}:s}(S) \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\}) &= O_{\mathsf{test}:s:n}(S'') \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\}) \;, \\
O_{\mathsf{write}:s}(S) \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\}) &= O_{\mathsf{write}:s:n}(S'') \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\}) \;, \\
O_{\mathsf{mover}}(S) \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\}) &= O_{\mathsf{mover}}(S'') \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\}) \;, \\
O_{\mathsf{movel}}(S) \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\}) &= O_{\mathsf{movel}}(S'') \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\}) \;,
\end{aligned}$$

$$\begin{aligned}
O_{\mathsf{test}:s}(S)(\mathsf{head}) = n' &\text{ iff } O_{\mathsf{test}:s:n}(S'')(\mathsf{M_{hd}}[n']) = \mathsf{T} \;, \\
O_{\mathsf{write}:s}(S)(\mathsf{head}) = n' &\text{ iff } O_{\mathsf{write}:s:n}(S'')(\mathsf{M_{hd}}[n']) = \mathsf{T} \;, \\
O_{\mathsf{mover}}(S)(\mathsf{head}) = n' &\text{ iff } O_{\mathsf{mover}}(S'')(\mathsf{M_{hd}}[n']) = \mathsf{T} \;, \\
O_{\mathsf{movel}}(S)(\mathsf{head}) = n' &\text{ iff } O_{\mathsf{movel}}(S'')(\mathsf{M_{hd}}[n']) = \mathsf{T} \;.
\end{aligned}$$

Let $(\|_{\mathrm{f}}(\langle CT_0 \rangle), S''_0) \vdash^*_{M''_T} (\|_{\mathrm{f}}(\langle p_1 \rangle \frown \ldots \frown \langle p_l \rangle), S'')$. Then we have for all $i, j \in [1, l]$, $n \in \mathbb{N}$ and $s \in \mathsf{B_{tape}}$:

(1) (a) if $p_i \in CT_n$ and $p_j \in CT_n$, then $i = j$,
    (b) $\mathsf{nt}(n+1) \circ CT'_n \in p_i$ iff $n = \max(\{m \mid \exists k \in [1, l] \bullet p_k \in CT_m\})$;
(2) if $S''(\mathsf{M_{hd}}[n]) = \mathsf{T}$, then:
    (a) there exists a unique $k \in [1, l]$ such that $p_k \in CT_n$,
    (b) for all $k' \in [1, l]$ and $n' \in \mathbb{N}$, $n' \neq n$ implies $p_{k'} \notin CT'_{n'}$;
(3) if $S''(\mathsf{M_{hd}}[n]) = \mathsf{T}$, $p_i \in CT_n$, $p_i \not\equiv \mathsf{S}$ and $p_i \not\equiv \mathsf{D}$, then there exist a $T'' \in \mathcal{S}_{M''_T}$ and $p'_1 \in p_1, \ldots, p'_{i-1} \in p_{i-1}$ such that $(\|_{\mathrm{f}}(\langle p_1 \rangle \frown \ldots \frown \langle p_l \rangle), S'') \vdash^*_{M''_T} (\|_{\mathrm{f}}(\langle p_i \rangle \frown \ldots \frown \langle p_l \rangle \frown \langle p'_1 \rangle \frown \ldots \frown \langle p'_{i-1} \rangle), T'')$ and $T'' \restriction \mathsf{M_{tape}} = S'' \restriction \mathsf{M_{tape}}$ and $T''(\mathsf{M_{hd}}[n]) = \mathsf{T}$;
(4) if $S''(\mathsf{M_{hd}}[n]) = \mathsf{T}$ and $p_1 \equiv CT_n$, then there exists a $T'' \in \mathcal{S}_{M''_T}$ such that $(\|_{\mathrm{f}}(\langle p_1 \rangle \frown \ldots \frown \langle p_l \rangle), S'') \vdash^*_{M''_T} (\|_{\mathrm{f}}(\langle CT_{n+1} \rangle \frown \langle CT'_n \rangle \frown \langle p_2 \rangle \frown \ldots \frown \langle p_l \rangle), T'')$ and $T'' \restriction \mathsf{M_{tape}} = S'' \restriction \mathsf{M_{tape}}$ and $T''(\mathsf{M_{hd}}[n]) = \mathsf{T}$;

(5) if $S''(\mathsf{M_{hd}}[n]) = \mathsf{T}$, $p_1 \equiv CT'_n$ and $S''(\mathsf{M_{thr}}[\mathit{ntl}(S'', S''(\mathsf{rr}))]) \notin \{\mathsf{S}, \mathsf{D}\}$, then:
   (a) if $\pi_2(S''(\mathsf{M_{thr}}[\mathit{ntl}(S'', S''(\mathsf{rr}))])) = \mathsf{test}{:}s$, then there exists a $T'' \in \mathcal{S}_{M''_T}$ such that $(\|_{\mathsf{f}}(\langle p_1\rangle \frown \ldots \frown \langle p_l\rangle), S'') \vdash^*_{M''_T} (\|_{\mathsf{f}}(\langle CT''_n\rangle \frown \langle p_2\rangle \frown \ldots \frown \langle p_l\rangle), T'')$ and $T'' \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\}) = O_{\mathsf{test}:s:n}(S'') \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\})$ and $T''(\mathsf{M_{hd}}[n]) = \mathsf{T}$;
   (b) if $\pi_2(S''(\mathsf{M_{thr}}[\mathit{ntl}(S'', S''(\mathsf{rr}))])) = \mathsf{write}{:}s$, then there exists a $T'' \in \mathcal{S}_{M''_T}$ such that $(\|_{\mathsf{f}}(\langle p_1\rangle \frown \ldots \frown \langle p_l\rangle), S'') \vdash^*_{M''_T} (\|_{\mathsf{f}}(\langle CT''_n\rangle \frown \langle p_2\rangle \frown \ldots \frown \langle p_l\rangle), T'')$ and $T'' \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\}) = O_{\mathsf{write}:s:n}(S'') \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\})$ and $T''(\mathsf{M_{hd}}[n]) = \mathsf{T}$;
   (c) if $\pi_2(S''(\mathsf{M_{thr}}[\mathit{ntl}(S'', S''(\mathsf{rr}))])) = \mathsf{mover}$, then there exist a $T'' \in \mathcal{S}_{M''_T}$ and $p'_2 \in p_2, \ldots, p'_l \in p_l$ such that $(\|_{\mathsf{f}}(\langle p_1\rangle \frown \ldots \frown \langle p_l\rangle), S'') \vdash^*_{M''_T} (\|_{\mathsf{f}}(\langle p'_2\rangle \frown \ldots \frown \langle p'_l\rangle \frown \langle CT''_n\rangle), T'')$ and $T'' \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\}) = O_{\mathsf{mover}}(S'') \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\})$ and $T''(\mathsf{M_{hd}}[n+1]) = \mathsf{T}$;
   (d) if $\pi_2(S''(\mathsf{M_{thr}}[\mathit{ntl}(S'', S''(\mathsf{rr}))])) = \mathsf{movel}$, then there exist a $T'' \in \mathcal{S}_{M''_T}$ and $p'_2 \in p_2, \ldots, p'_l \in p_l$ such that $(\|_{\mathsf{f}}(\langle p_1\rangle \frown \ldots \frown \langle p_l\rangle), S'') \vdash^*_{M''_T} (\|_{\mathsf{f}}(\langle p'_2\rangle \frown \ldots \frown \langle p'_l\rangle \frown \langle CT''_n\rangle), T'')$ and $T'' \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\}) = O_{\mathsf{movel}}(S'') \restriction (\mathsf{M_{tape}} \cup \{\mathsf{rr}\})$ and either $n > 0$ and $T''(\mathsf{M_{hd}}[n-1]) = \mathsf{T}$ or $n = 0$ and $T''(\mathsf{M_{hd}}[n]) = \mathsf{T}$;

(6) if $S''(\mathsf{M_{hd}}[n]) = \mathsf{T}$, $p_1 \equiv CT'_n$ and $S''(\mathsf{M_{thr}}[\mathit{ntl}(S'', S''(\mathsf{rr}))]) \in \{\mathsf{S}, \mathsf{D}\}$, then:
   (a) if $S''(\mathsf{M_{thr}}[\mathit{ntl}(S'', S''(\mathsf{rr}))]) = \mathsf{S}$, then there exists a $T'' \in \mathcal{S}_{M''_T}$ such that $(\|_{\mathsf{f}}(\langle p_1\rangle \frown \ldots \frown \langle p_l\rangle), S'') \vdash^*_{M''_T} (\mathsf{S}, T'')$ and $T'' \restriction \mathsf{M_{tape}} = S'' \restriction \mathsf{M_{tape}}$ and $T''(\mathsf{M_{hd}}[n]) = \mathsf{T}$;
   (b) if $S''(\mathsf{M_{thr}}[\mathit{ntl}(S'', S''(\mathsf{rr}))]) = \mathsf{D}$, then there exists a $T'' \in \mathcal{S}_{M''_T}$ such that $(\|_{\mathsf{f}}(\langle p_1\rangle \frown \ldots \frown \langle p_l\rangle), S'') \vdash^*_{M''_T} (\mathsf{D}, T'')$ and $T'' \restriction \mathsf{M_{tape}} = S'' \restriction \mathsf{M_{tape}}$ and $T''(\mathsf{M_{hd}}[n]) = \mathsf{T}$;

(7) if $S''(\mathsf{M_{hd}}[n]) = \mathsf{T}$ and $p_1 \equiv CT''_n$, then there exist a $T'' \in \mathcal{S}_{M''_T}$ and $p'_2 \in p_2, \ldots, p'_l \in p_l$ such that $(\|_{\mathsf{f}}(\langle p_1\rangle \frown \ldots \frown \langle p_l\rangle), S'') \vdash^*_{M''_T} (\|_{\mathsf{f}}(\langle CT'_n\rangle \frown \langle p'_2\rangle \frown \ldots \frown \langle p'_l\rangle), T'')$ and $T'' \restriction \mathsf{M_{tape}} = S'' \restriction \mathsf{M_{tape}}$ and $T''(\mathsf{M_{hd}}[n]) = \mathsf{T}$.

Property 1 is easily proved by induction on m. Using property 1, property 2 is easily proved by induction on n. Using properties 1 and 2, property 3 is easily proved by case distinction between the different forms $p_1, \ldots, p_l$ can take. Using properties 1 and 2, the remaining properties are easily proved by case distinction between the different forms $p_2, \ldots, p_l$ can take.

Let (p_m, S_m) be the m+1-th element in the full path of (p, S_0) on M_T, let (p''_0, S''_0) be the first element in the full path of $(\|_{\mathsf{f}}(\langle CT_0\rangle), S''_0)$ on M''_T, and let (p''_{m+1}, S''_{m+1}) be the element in the full path of $(\|_{\mathsf{f}}(\langle CT_0\rangle), S''_0)$ on M''_T that follows the m+1-th element of which the first component equals $\|_{\mathsf{f}}(\langle \mathsf{exsba}{:}n \circ CT''_n\rangle \frown \alpha)$ for some $n \in \mathbb{N}$ and $\alpha \in \mathcal{T}_{\mathsf{finrec}}{}^*$. Then, using the above equations and other properties, it is straightforward to prove by induction on m that:

- p_m is represented by the part of $\mathsf{s}_{\mathsf{thr}}(p)$ to which $\mathit{ntl}(S''_m, S''_m(\mathsf{rr}))$ points;
- $S_m \restriction (\mathsf{M}_{\mathsf{tape}} \cup \{\mathsf{rr}\}) = S''_m \restriction (\mathsf{M}_{\mathsf{tape}} \cup \{\mathsf{rr}\})$ and for all $n \in \mathbb{N}$, $S_m(\mathsf{head}) = n$ iff $S''_m(\mathsf{M}_{\mathsf{hd}}[n]) = \mathsf{T}$.

(if $m < \|(p, S_0)\|_{M_T}$ in case p converges from S_0 on M_T). From this, the theorem follows immediately. □

Example 7 *Consider again the Turing machine from Example 1. The Turing thread that corresponds to the finite-state control of this Turing machine is given in Example 3. The stored graph representation* s_{thr} *of this Turing thread is as follows:*

$$\begin{aligned}
&s_{\mathsf{thr}}(\mathsf{M}_{\mathsf{thr}}[0]) = (1, \mathsf{test{:}}0, 2)\ ,\\
&s_{\mathsf{thr}}(\mathsf{M}_{\mathsf{thr}}[1]) = (5, \mathsf{write{:}}1, 5)\ ,\\
&s_{\mathsf{thr}}(\mathsf{M}_{\mathsf{thr}}[2]) = (3, \mathsf{test{:}}1, 4)\ ,\\
&s_{\mathsf{thr}}(\mathsf{M}_{\mathsf{thr}}[3]) = (5, \mathsf{write{:}}0, 5)\ ,\\
&s_{\mathsf{thr}}(\mathsf{M}_{\mathsf{thr}}[4]) = \mathsf{S}\ ,\\
&s_{\mathsf{thr}}(\mathsf{M}_{\mathsf{thr}}[5]) = (6, \mathsf{test{:}}0, 7)\ ,\\
&s_{\mathsf{thr}}(\mathsf{M}_{\mathsf{thr}}[6]) = (0, \mathsf{mover}, 0)\ ,\\
&s_{\mathsf{thr}}(\mathsf{M}_{\mathsf{thr}}[7]) = (6, \mathsf{test{:}}1, 6)\ .
\end{aligned}$$

Supposing that execution of this stored thread started off under control of the multi-thread $\|_{\mathrm{f}}(\langle CT_0\rangle)$ *and* n *is the head position, execution of basic action* $\mathsf{test{:}}s$ *amounts to performing operation* $O_{\mathsf{test}:s:n}$ *and execution of basic action* $\mathsf{write{:}}s$ *amounts to performing operation* $O_{\mathsf{write}:s:n}$*. Thus, the adaptation of the Turing thread as described in Section 9 is in fact carried out in a dynamic manner.*

The way to simulate Turing machines on a Maurer machine described in this section involves interleaving of the threads in a thread vector. The thread vector concerned consists of finite-state threads only. Initially, the thread vector consists of one thread. The length of the thread vector usually increases during execution. However, we conclude from Theorem 6 and the definition of the apply operator that it remains finite in the case of convergence.

In Section 8, the operations used to simulate Turing machines includes operations with an infinite input region and operations with an infinite output region. In Section 9, only operations with a finite input region and a finite output region are used together with adapted Turing threads. However, another kind of infinity arises: the adaptation turns many Turing threads, which are finite-state threads, into infinite-state threads. In this section, the adaptation of Turing threads is circumvented by storing the Turing threads and then executing the stored Turing threads under control of a multi-thread that makes the head position part of the operations. By using thread forking in the way described, the execution remains controlled by a finite-state thread in the case of convergence. However, still another kind of infinity arises: the thread forking function needed is an injective function with an infinite domain, viz. $\mathbb{N}$.

11 Fair Strategic Interleaving

Cyclic interleaving with perfect forking is a simple instance of a fair interleaving strategy. In this section, we make precise what it means for basic interleaving strategies with support of perfect forking to be fair. It happens that the way to simulate Turing machines on a Maurer machine presented in Section 10 works with any fair basic interleaving strategy with support of perfect forking.

In [11], it is demonstrated that it is in essence open-ended what counts as an interleaving strategy. However, here we have to make precise what we consider to be an interleaving strategy. Our choice is conditioned by the simple fact that strategies that are not relevant to the present purpose can be left out. This means that we consider only basic interleaving strategies with support of perfect forking, but without support of other special features. For instance, we do not consider interleaving strategies that support the case where processing of certain actions may be temporarily blocked and/or blocked forever. And we do not consider any kind of non-perfect forking either.

A *basic strategic interleaving* operator $\|_s(\_)$ is an operator on a thread vector such that for all $\alpha \in \mathcal{T}_{\mathsf{finrec}}{}^*$, $p', p'' \in \mathcal{T}_{\mathsf{finrec}}$, $a \in \mathcal{A}_{\mathsf{tau}} \setminus \mathcal{NT}$ and $n \in \mathrm{dom}(\phi)$:

$$
\begin{aligned}
&\|_s(\langle\,\rangle) = \mathsf{S}\;,\\
&\|_s(\langle \mathsf{S}\rangle \curvearrowright \alpha) = \|_s(\alpha)\;,\\
&\|_s(\langle \mathsf{D}\rangle \curvearrowright \alpha) = \mathsf{S}_{\mathsf{D}}(\|_s(\alpha))\;,\\
&\exists! \alpha', \alpha'' \in \mathcal{T}_{\mathsf{finrec}}{}^* \bullet\\
&\quad (\alpha' \in \mathrm{perm}(\langle p'\rangle \curvearrowright \alpha) \land \alpha'' \in \mathrm{perm}(\langle p''\rangle \curvearrowright \alpha) \land\\
&\quad\;\; \|_s(\langle p' \unlhd a \unrhd p''\rangle \curvearrowright \alpha) = \|_s(\alpha') \unlhd a \unrhd \|_s(\alpha''))\;,\\
&\exists! \alpha' \in \mathcal{T}_{\mathsf{finrec}}{}^* \bullet\\
&\quad (\alpha' \in \mathrm{perm}(\langle p'\rangle \curvearrowright \alpha \curvearrowright \langle \phi(n)\rangle) \land\\
&\quad\;\; \|_s(\langle p' \unlhd \mathsf{nt}(n) \unrhd p''\rangle \curvearrowright \alpha) = \mathsf{tau} \circ \|_s(\alpha'))\;.^{5}
\end{aligned}
$$

[5] We write D^* for the set of all finite sequences with elements from set D, D^+ for the set of all non-empty finite sequences with elements from set D, and $\mathrm{perm}(\alpha)$ for

The strategic interleaving operators characterized here basically operate as follows: at each interleaving step, the first thread in the thread vector gets a turn to perform an action and then the remaining thread vector is permuted in a deterministic manner. Hence, for a given basic strategic interleaving operator $\|_s(\_)$, the axioms can always be given in the following way:

$$
\begin{aligned}
&\|_s(\langle\,\rangle) = \mathsf{S}\;,\\
&\|_s(\langle \mathsf{S}\rangle \curvearrowright \alpha) = \|_s(\alpha)\;,\\
&\|_s(\langle \mathsf{D}\rangle \curvearrowright \alpha) = \mathsf{S}_{\mathsf{D}}(\|_s(\alpha))\;,\\
&\|_s(\langle x \trianglelefteq a \trianglerighteq y\rangle \curvearrowright \alpha) = \|_s(pv_s^{+a}(\langle x\rangle \curvearrowright \alpha)) \trianglelefteq a \trianglerighteq \|_s(pv_s^{-a}(\langle y\rangle \curvearrowright \alpha))\;,\\
&\|_s(\langle x \trianglelefteq \mathsf{nt}(n) \trianglerighteq y\rangle \curvearrowright \alpha) = \mathsf{tau} \circ \|_s(pv_s^{+\mathsf{nt}(n)}(\langle x\rangle \curvearrowright \alpha \curvearrowright \langle \phi(n)\rangle))\;,
\end{aligned}
$$

where a stands for an arbitrary member of $\mathcal{A}_{\mathsf{tau}} \backslash \mathcal{NT}$, for unary functions pv_s^{+a}, pv_s^{-a} and $pv_s^{+\mathsf{nt}(n)}$ on $\mathcal{T}_{\mathsf{finrec}}{}^{+}$ such that, for all $\alpha \in \mathcal{T}_{\mathsf{finrec}}{}^{+}$, $pv_s^{+a}(\alpha) \in \mathrm{perm}(\alpha)$, $pv_s^{-a}(\alpha) \in \mathrm{perm}(\alpha)$ and $pv_s^{+\mathsf{nt}(n)}(\alpha) \in \mathrm{perm}(\alpha)$.

In order to determine whether a basic interleaving strategy is fair, we need to know how thread vectors are permuted. If α is a thread vector in which a thread occurs more than once, we cannot infer from α and the thread vector resulting from a permutation of α how α is permuted. Hence, the functions pv_s^{+a} and pv_s^{-a} are not sufficient to determine whether a basic interleaving strategy is fair.

Therefore, we assume that, for all $a \in \mathcal{A}_{\mathsf{tau}}$, $b \in \mathcal{A} \setminus \mathcal{NT}$ and $\alpha \in \mathcal{T}_{\mathsf{finrec}}{}^{+}$, functions $pp_s^{+a}(\alpha), pp_s^{-b}(\alpha) : [1,|\alpha|] \to [1,|\alpha|]$ are given such that:

$$
\begin{aligned}
&pv_s^{+a}(\langle p_1\rangle \curvearrowright \ldots \curvearrowright \langle p_n\rangle) = \langle p'_1\rangle \curvearrowright \ldots \curvearrowright \langle p'_n\rangle \;\Rightarrow\\
&\quad \forall i \in [1,n] \bullet p_i \equiv p'_{pp_s^{+a}(\langle p_1\rangle \curvearrowright \ldots \curvearrowright \langle p_n\rangle)(i)}\;,\\
&pv_s^{-b}(\langle p_1\rangle \curvearrowright \ldots \curvearrowright \langle p_n\rangle) = \langle p'_1\rangle \curvearrowright \ldots \curvearrowright \langle p'_n\rangle \;\Rightarrow\\
&\quad \forall i \in [1,n] \bullet p_i \equiv p'_{pp_s^{-b}(\langle p_1\rangle \curvearrowright \ldots \curvearrowright \langle p_n\rangle)(i)}\;.
\end{aligned}
$$

Auxiliary relations $\_ \overset{+a}{\leadsto} \_ \subseteq \mathcal{T}_{\mathsf{finrec}}{}^{+} \times \mathcal{T}_{\mathsf{finrec}}{}^{+}$, for $a \in \mathcal{A}_{\mathsf{tau}}$, and $\_ \overset{-b}{\leadsto} \_ \subseteq \mathcal{T}_{\mathsf{finrec}}{}^{+} \times \mathcal{T}_{\mathsf{finrec}}{}^{+}$, for $b \in \mathcal{A} \setminus \mathcal{NT}$, are used below to define fairness of basic

the set of all permutations of sequence α.

interleaving strategies. They are defined as follows:

$$
\begin{array}{l}
\alpha \overset{+a}{\rightsquigarrow} \alpha' \Leftrightarrow \\
\quad \exists p, p' \in \mathcal{T}_{\mathsf{finrec}}, \alpha'' \in {\mathcal{T}_{\mathsf{finrec}}}^{*} \bullet \\
\qquad (\alpha = \langle p \unlhd a \unrhd p' \rangle \frown \alpha'' \land \alpha' = pv_s^{+a}(\langle p \rangle \frown \alpha'')) \qquad \text{if } a \notin \mathcal{NT} \;, \\[1ex]
\alpha \overset{+\mathsf{nt}(n)}{\rightsquigarrow} \alpha' \Leftrightarrow \\
\quad \exists p, p' \in \mathcal{T}_{\mathsf{finrec}}, \alpha'' \in {\mathcal{T}_{\mathsf{finrec}}}^{*} \bullet \\
\qquad (\alpha = \langle p \unlhd \mathsf{nt}(n) \unrhd p' \rangle \frown \alpha'' \land \alpha' = pv_s^{+\mathsf{nt}(n)}(\langle p \rangle \frown \alpha'' \frown \langle \phi(n) \rangle)) \;, \\[1ex]
\alpha \overset{-b}{\rightsquigarrow} \alpha' \Leftrightarrow \\
\quad \exists p, p' \in \mathcal{T}_{\mathsf{finrec}}, \alpha'' \in {\mathcal{T}_{\mathsf{finrec}}}^{*} \bullet \\
\qquad (\alpha = \langle p \unlhd b \unrhd p' \rangle \frown \alpha'' \land \alpha' = pv_s^{-b}(\langle p' \rangle \frown \alpha'')) \;.
\end{array}
$$

In other words, $\alpha \overset{+a}{\rightsquigarrow} \alpha'$ iff $\|_s(\alpha)$ is capable of performing basic action a and then proceeding as $\|_s(\alpha')$ if a positive reply is produced, and similarly for $\alpha \overset{-b}{\rightsquigarrow} \alpha'$.

Let $\|_s(\_)$ be a basic strategic interleaving operator. Then $\|_s(\_)$ is *fair* if for all $\alpha_0, \alpha_1, \ldots \in {\mathcal{T}_{\mathsf{finrec}}}^{+}$ and $j_0 \in [1, |\alpha_0|]$, $j_1 \in [1, |\alpha_1|]$, ... :

$$
\begin{array}{l}
\bigwedge\limits_{i \in \mathbb{N}} (\exists a \in \mathcal{A}_{\mathsf{tau}} \bullet (\alpha_i \overset{+a}{\rightsquigarrow} \alpha_{i+1} \land pp_s^{+a}(\alpha_i)(j_i) = j_{i+1}) \lor \\
\quad \exists a \in \mathcal{A} \setminus \mathcal{NT} \bullet (\alpha_i \overset{-a}{\rightsquigarrow} \alpha_{i+1} \land pp_s^{-a}(\alpha_i)(j_i) = j_{i+1})) \Rightarrow \bigvee\limits_{i \in \mathbb{N}} j_{i+1} = 1 \;.
\end{array}
$$

In words, $\|_s(\_)$ is fair if, for each thread vector that leads to infinitely many interleaving steps, there will eventually come a next turn for each thread in that thread vector.

According to the above definitions, cyclic interleaving with perfect forking is a fair basic interleaving strategy. The way to simulate Turing machines on a Maurer machine shown in Section 10 works not only with cyclic interleaving, but also with any other fair basic interleaving strategy.

Theorem 8 *Theorem 6 goes through if we replace the strategic interleaving operator for cyclic interleaving by any other fair basic interleaving strategy.*

PROOF. The proof follows the same line as the proof of Theorem 6. Properties 4, 5a and 5b from that proof are too strong in the case of an arbitrary fair basic interleaving strategy. We have the following weaker properties instead:

4′. if $S''(\mathsf{M}_{\mathsf{hd}}[n]) = \mathsf{T}$ and $p_1 \equiv CT_n$, then there exist a $T'' \in \mathcal{S}_{M''_T}$ and

$p'_2 \in p_2, \ldots, p'_l \in p_l$ such that $(\|_{\mathrm{f}}(\langle p_1 \rangle \curvearrowright \ldots \curvearrowright \langle p_l \rangle), S'') \vdash^*_{M''_T} (\|_{\mathrm{f}}(\langle CT_{n+1} \rangle \curvearrowright \langle CT'_n \rangle \curvearrowright \langle p'_2 \rangle \curvearrowright \ldots \curvearrowright \langle p'_l \rangle), T'')$ and $T'' \restriction \mathsf{M}_{\mathsf{tape}} = S'' \restriction \mathsf{M}_{\mathsf{tape}}$ and $T''(\mathsf{M}_{\mathsf{hd}}[n]) = \mathsf{T}$;

5′. if $S''(\mathsf{M}_{\mathsf{hd}}[n]) = \mathsf{T}$, $p_1 \equiv CT'_n$ and $S''(\mathsf{M}_{\mathsf{thr}}[ntl(S'', S''(\mathsf{rr}))]) \notin \{\mathsf{S}, \mathsf{D}\}$, then:
   (a) if $\pi_2(S''(\mathsf{M}_{\mathsf{thr}}[ntl(S'', S''(\mathsf{rr}))])) = \mathsf{test}{:}s$, then there exist a $T'' \in \mathcal{S}_{M''_T}$ and $p'_2 \in p_2, \ldots, p'_l \in p_l$ such that $(\|_{\mathrm{f}}(\langle p_1 \rangle \curvearrowright \ldots \curvearrowright \langle p_l \rangle), S'') \vdash^*_{M''_T} (\|_{\mathrm{f}}(\langle CT''_n \rangle \curvearrowright \langle p'_2 \rangle \curvearrowright \ldots \curvearrowright \langle p'_l \rangle), T'')$ and $T'' \restriction (\mathsf{M}_{\mathsf{tape}} \cup \{\mathsf{rr}\}) = O_{\mathsf{test}:s:n}(S'') \restriction (\mathsf{M}_{\mathsf{tape}} \cup \{\mathsf{rr}\})$ and $T''(\mathsf{M}_{\mathsf{hd}}[n]) = \mathsf{T}$;
   (b) if $\pi_2(S''(\mathsf{M}_{\mathsf{thr}}[ntl(S'', S''(\mathsf{rr}))])) = \mathsf{write}{:}s$, then there exist a $T'' \in \mathcal{S}_{M''_T}$ and $p'_2 \in p_2, \ldots, p'_l \in p_l$ such that $(\|_{\mathrm{f}}(\langle p_1 \rangle \curvearrowright \ldots \curvearrowright \langle p_l \rangle), S'') \vdash^*_{M''_T} (\|_{\mathrm{f}}(\langle CT''_n \rangle \curvearrowright \langle p'_2 \rangle \curvearrowright \ldots \curvearrowright \langle p'_l \rangle), T'')$ and $T'' \restriction (\mathsf{M}_{\mathsf{tape}} \cup \{\mathsf{rr}\}) = O_{\mathsf{write}:s:n}(S'') \restriction (\mathsf{M}_{\mathsf{tape}} \cup \{\mathsf{rr}\})$ and $T''(\mathsf{M}_{\mathsf{hd}}[n]) = \mathsf{T}$.

These weaker properties are sufficient to complete the proof. □

12 Concluding Remarks

There are many ways to simulate Turing machines on Maurer machines. In this paper, we have presented three ways which give insight into the connections between Turing machines, Maurer machines and real computers:

- In the first way, the Maurer machine on which Turing machines are simulated has the most obvious operations for the simulation of Turing machines. Moreover, the transition function of the Turing machine in question is rendered in an obvious way into a finite-state thread which is applied to that Maurer machine. Unlike real computers, the Maurer machine used for the simulation has operations with an infinite input region or an infinite output region.
- In the second way, the Maurer machine on which Turing machines are simulated has only operations with a finite input region and a finite output region. This is attained by replacing each operation with an infinite input region or an infinite output region by a countably infinite number of operations, namely one for each different head position. The necessary adaptation of the thread into which the transition function of a Turing machine is rendered, usually results in an infinite-state thread. Unlike finite-state threads, infinite-state threads cannot be regarded as behaviours of programs under execution on a real computer.
- In the third way, the Maurer machine on which Turing machines are simulated has again only operations with a finite input region and a finite output region. However, the thread into which the transition function of a Turing machine is rendered is not adapted, but first stored in the memory of the

Maurer machine and then executed under control of a multi-thread that makes the head position part of the operations. The multi-thread forks off for every head position a control thread of itself on the head reaching the preceding position for the first time. Thus, the multi-thread remains finite in the case of convergence.

The third way also illustrates that some main concepts of contemporary programming, namely multi-threads and thread forking, have an interesting theoretical application.

In [10], we have demonstrated the feasibility of an approach based on Maurer machines and basic thread algebra to model micro-architectures and to verify their correctness and anticipated speed-up results. In [4], we have made use of the experience gained in that feasibility study to model micro-architectures with pipelined instruction processing. Maurer's model for computers is relatively unknown, whereas Turing's model, which is quite different, belongs to the foundations of theoretical computer science. To relate our approach to model and analyse micro-architectures to these foundations, we have investigated the connections between the two models in this paper.

The work presented in this paper, as well as the work presented in [10,4], was in part carried out in the framework of a project investigating micro-threading [13,19], 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 forking. In [6], we have investigated parallelization for simple programs, called straight-line programs, using Maurer machines and basic thread algebra as well.

In [4,6], program algebra [3] is used, in addition to Maurer machines and basic thread algebra, to investigate issues related to instruction processing. This is convenient because programs are viewed as instruction sequences in program algebra. In this paper, program algebra is not used. Only threads matter to the simulation of Turing machines on Maurer machines, in the sense that only the threads represented by the programs that could replace them would be relevant. The replacement would lead to needless complications when investigating the simulation of Turing machines on Maurer machines.

The work presented in this paper, as well as the work presented in [10,4,6], is an application of thread algebra. Thread algebra is the theory about threads and multi-threads, introduced in [11], which originates in basic thread algebra. Extensions of the theory introduced in [11] are presented in [7–9].

The work presented in this paper, as well as the work presented in [10,4], has convinced us that a special notation for the description of Maurer machines 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. We aim at a notation of which the semantics can simply be given by a translation to logical formulas, much in the spirit of predicative methodology [15].

Acknowledgements

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

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

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

References

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

[2] J. A. Bergstra and I. Bethke. Polarized process algebra and program equivalence. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Proceedings 30th ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 1–21. Springer-Verlag, 2003.

[3] J. A. Bergstra and M. E. Loots. Program algebra for sequential code. *Journal of Logic and Algebraic Programming*, 51(2):125–156, 2002.

[4] J. A. Bergstra and C. A. Middelburg. Maurer computers for pipelined instruction processing. Computer Science Report 06-12, Department of Mathematics and Computer Science, Eindhoven University of Technology, March 2006.

[5] J. A. Bergstra and C. A. Middelburg. Splitting bisimulations and retrospective conditions. *Information and Computation*, 204(7):1083–1138, 2006.

[6] J. A. Bergstra and C. A. Middelburg. Synchronous cooperation for explicit multi-threading. Computer Science Report 06-29, Department of Mathematics and Computer Science, Eindhoven University of Technology, September 2006.

[7] J. A. Bergstra and C. A. Middelburg. Thread algebra with multi-level strategies. *Fundamenta Informaticae*, 71(2/3):153–182, 2006.

[8] J. A. Bergstra and C. A. Middelburg. A thread calculus with molecular dynamics. Computer Science Report 06-24, Department of Mathematics and Computer Science, Eindhoven University of Technology, August 2006.

[9] J. A. Bergstra and C. A. Middelburg. Distributed strategic interleaving with load balancing. Computer Science Report 07-03, Department of Mathematics and Computer Science, Eindhoven University of Technology, January 2007.

[10] J. A. Bergstra and C. A. Middelburg. Maurer computers with single-thread control. To appear in *Fundamenta Informaticae*, 2007. Preliminary version: Computer Science Report 05-17, Department of Mathematics and Computer Science, Eindhoven University of Technology.

[11] J. A. Bergstra and C. A. Middelburg. 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.

[12] J. A. Bergstra and A. Ponse. Combining programs and state machines. *Journal of Logic and Algebraic Programming*, 51(2):175–192, 2002.

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

[14] M. D. Davis and E. J. Weyuker. *Computability, Complexity, and Languages.* Academic Press, New York, 1983.

[15] E. C. R. Hehner, L. E. Gupta, and A. J. Malton. Predicative methodology. *Acta Informatica*, 23:487–505, 1986.

[16] H. Hermes. *Enumerability, Decidability, Computability.* Springer-Verlag, Berlin, 1965.

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

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

[19] C. R. Jesshope and B. Luo. Micro-threading: A new approach to future RISC. In *Australian Computer Architecture Conference 2000*, pages 34–41. IEEE Computer Society Press, 2000.

[20] P. Linz. *An Introduction to Formal Languages and Automata.* Jones and Bartlett Publishers, Sudbury, MA, 1997.

[21] W. D. Maurer. A theory of computer instructions. *Journal of the ACM*, 13(2):226–235, 1966.

[22] W. D. Maurer. A theory of computer instructions. *Science of Computer Programming*, 60:244–273, 2006.

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

[24] A. M. Turing. On computable numbers, with an application to the Entscheidungs problem. *Proceedings of the London Mathematical Society, Series 2*, 42:230–265, 1937. Correction: *ibid*, 43:544–546, 1937.