Simulating Turing machines on Maurer machines

Citation for published version (APA):

Document status and date:
Published: 01/01/2005

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Simulating Turing Machines on Maurer Machines

J.A. Bergstra\textsuperscript{1,2} and C.A. Middelburg\textsuperscript{3}

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

Abstract. Maurer machines are much closer to real computers than Turing machines. Computer instructions play a prominent part in Maurer machines. We show a straightforward way to simulate Turing machines on a Maurer machine. It illustrates that the instruction implicitly executed by a Turing machine on a test or write step 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. Real computers do not have such instructions. We show two ways to get rid of this kind of infinity.

Keywords: Turing machines, Maurer machines, thread algebra, stored threads, strategic interleaving, thread forking, fair interleaving strategies.


1 Introduction

In [12], Maurer proposes a model for computers that is quite different from the well-known models such as register machines, multistack machines and Turing machines (see e.g. [9]). The strength of Maurer’s model is that it is close to real computers. Computer instructions play a prominent part in his model.

In [4], we developed a theory which covers basic issues concerning stored threads and their execution. 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 of the stored thread correspond to instructions of the Maurer machine.

To describe threads, we used an extension of BTA (Basic Thread Algebra), introduced in [2] under the name BPPA (Basic Polarized Process Algebra). BTA is a form of process algebra which is tailored to the description of the behaviour of deterministic sequential programs under execution. The extension 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 a sequence of state changes according to the instructions that the Maurer machine associates with the basic actions performed by the thread.

In this paper, we show a straightforward way to simulate Turing machines on a Maurer machine. It illustrates that the instruction implicitly executed by a Turing machine on a test or write step 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 instruction has an infinite input region and a write instruction has an infinite output region. Real computers do not have such instructions. We show also that the instructions concerned can be replaced by instructions with a finite input region and a finite output region if we abandon the restriction to machines with a finite-state control.

In the straightforward way to simulate Turing machines on a Maurer machine, a thread applied to the Maurer machine corresponds to the finite-state control of the Turing machine in question. This thread is definable by a finite recursive specification. However, the adaptation of this way to simulate Turing machines to the use of instructions 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, in the simulation of Turing machines just as well, one kind of infinity has been replaced by another kind. We show a way to get around the latter kind of infinity in the case of convergent computations. 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 instructions and (b) the controlling thread grows, by means of thread forking, whenever a head position occurs for the first time. For this, the operator for interleaving of threads from [4] is adapted to thread forking. This operator is based on the simplest deterministic interleaving strategy, namely cyclic interleaving. In [3], 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 BTA 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 instructions (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 a simple 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 instructions with a finite input region and a finite output region only (Sections 9 and 10). After that, we discuss why the latter way, which uses cyclic interleaving, works for any fair interleaving strategy (Section 11). Finally, we make some concluding remarks (Section 12).
2 Maurer Computers

In this section, we shortly review Maurer computers, i.e. computers as defined by Maurer in [12].

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

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

and satisfies the following conditions:

- if $S_1, S_2 \in S$, $M' \subseteq M$ 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 $S_3 \in S$;
- if $S_1, S_2 \in S$, 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 $S$ are called the states, and the members of $\mathcal{I}$ are called the instructions. It is obvious that the first condition is satisfied if $C$ is complete, i.e. if $S$ 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.

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

$$IR(I) = \{x \in M \mid \exists S_1, S_2 \in S, y \in OR(I) \cdot \forall z \in M \setminus \{x\} \cdot S_1(z) = S_2(z) \land I(S_1)(y) \neq I(S_2)(y)\},$$

$$OR(I) = \{x \in M \mid \exists S \in S \cdot S(x) \neq I(S)(x)\}.$$

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

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

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

$$AR(M'', I) = \{x \in OR(I) \mid \exists S_1, S_2 \in S \cdot \forall z \in IR(I) \setminus M'' \cdot S_1(z) = S_2(z) \land I(S_1)(x) \neq I(S_2)(x)\}.$$

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

In [12], Maurer gives many results about the relation between the input region and output region of instructions, the composition of instructions, the decomposition of instructions and the existence of instructions. In [4], we summarize the main results.
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 \( \tau \not\in \mathcal{A} \). We write \( \mathcal{A}_\tau \) for \( \mathcal{A} \cup \{ \tau \} \). BTA has the following constants and operators:

- the deadlock constant \( D \);
- the termination constant \( S \);
- for each \( a \in \mathcal{A}_\tau \), a binary postconditional composition operator \( \triangleright a \triangleright \).

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 \( p \triangleright a \triangleright p \).

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 \( p \triangleright a \triangleright q \) will proceed as \( p \) if the processing of \( a \) leads to the reply \( T \) (called a positive reply), and it will proceed as \( q \) if the processing of \( a \) leads to the reply \( F \) (called a negative reply). 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: \( x \triangleright \tau \triangleright y = x \triangleright \tau \triangleright x \).

A recursive specification over BTA is a set of equations \( E = \{ X = t \mid X \in V \} \), where \( V \) is a set of variables and each \( t \) is a term of BTA that only contains variables from \( V \). We write \( V(E) \) 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 \( t' \triangleright a \triangleright t'' \) 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 [1, 2]. 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 \( X \in V(E) \), we add

\[
x \triangleleft \tau \triangleright y = x \triangleleft \tau \triangleright x \quad \text{T1}
\]

### Table 1. Axiom of BTA

<table>
<thead>
<tr>
<th>Axiom of BTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \triangleleft \tau \triangleright y = x \triangleleft \tau \triangleright x )</td>
</tr>
</tbody>
</table>

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 \( p \triangleleft a \triangleleft p \).

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 \( p \triangleleft a \triangleleft q \) will proceed as \( p \) if the processing of \( a \) leads to the reply \( T \) (called a positive reply), and it will proceed as \( q \) if the processing of \( a \) leads to the reply \( F \) (called a negative reply). 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: \( x \triangleleft \tau \triangleright y = x \triangleleft \tau \triangleright x \).

A recursive specification over BTA is a set of equations \( E = \{ X = t \mid X \in V \} \), where \( V \) is a set of variables and each \( t \) is a term of BTA that only contains variables from \( V \). We write \( V(E) \) 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 \( t' \triangleleft a \triangleleft t'' \) 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 [1, 2]. 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 \( X \in V(E) \), we add

\[
x \triangleleft \tau \triangleright y = x \triangleleft \tau \triangleright x \quad \text{T1}
\]
Table 2. Axioms for guarded recursion

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (X</td>
<td>E) = (tx</td>
</tr>
<tr>
<td>( E \Rightarrow X = (X</td>
<td>E) ) if ( X \in V(E) )</td>
</tr>
</tbody>
</table>

Table 3. Approximation induction principle

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \pi_0(x) = D )</td>
<td>P0</td>
</tr>
<tr>
<td>( \pi_{n+1}(S) = S )</td>
<td>P1</td>
</tr>
<tr>
<td>( \pi_{n+1}(D) = D )</td>
<td>P2</td>
</tr>
<tr>
<td>( \pi_{n+1}(x \sqsubseteq a \triangleright y) = \pi_n(x) \sqsubseteq a \triangleright \pi_n(y) )</td>
<td>P3</td>
</tr>
<tr>
<td>( (\bigwedge_{n \geq 0} \pi_n(x) = \pi_n(y)) \Rightarrow x = y )</td>
<td>AIP</td>
</tr>
</tbody>
</table>

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 \( (t|E) \) for \( t \) with, for all \( X \in V(E) \), all occurrences of \( X \) in \( t \) replaced by \( (X|E) \). We add the axioms for guarded recursion given in Table 2 to the axioms of BTA. In this table, \( t \), \( X \) 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 \( X \), \( t \) 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 \( (X|E) = (tx|E) \) for a fixed \( E \) express that the constants \( \langle X|E \rangle \) make up a solution of \( E \). The conditional equations \( E \Rightarrow X = (X|E) \) 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 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. Following [1, 2], approximation of depth \( n \) is phrased in terms of a unary projection operator \( \pi_n(\cdot) \). The projection operators are defined inductively by means of the axioms P0–P3 given in Table 3. In this table, \( a \) stands for an arbitrary member of \( \mathcal{A}_{\text{tau}} \). 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 [5, 4]. Henceforth, we write \( T_{\text{finrec}} \) for the set of all terms of BTA with recursion in which no constants \( \langle X|E \rangle \) for infinite \( E \) occur, and \( T_{\text{finrec}} \) 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 \( T_{\text{finrec}}(A) \), where \( A \subseteq \mathcal{A} \), for the set of all closed terms from \( T_{\text{finrec}} \).
Table 4. Axioms for cyclic interleaving with perfect forking

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$∥_I(()) = S$</td>
<td>CSIf1</td>
</tr>
<tr>
<td>$∥_I(S) \alpha = ∥_I(\alpha)$</td>
<td>CSIf2</td>
</tr>
<tr>
<td>$∥<em>I(D) \alpha = S</em>\alpha(∥_I(\alpha))$</td>
<td>CSIf3</td>
</tr>
<tr>
<td>$∥_I((x ≤ a \triangleright y) \alpha) = ∥_I((\alpha \triangleright (x)) ≤ a \triangleright ∥_I(\alpha \triangleright (y)))$</td>
<td>CSIf4</td>
</tr>
<tr>
<td>$∥_I((x ≤ n(t(n)) \triangleright y) \alpha) = \tau \circ ∥_I((\alpha \triangleright (\phi(n)) \triangleright (x)))$</td>
<td>CSIf5</td>
</tr>
</tbody>
</table>

Table 5. Axioms for deadlock at termination

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_D(S) = D$</td>
<td>S2D1</td>
</tr>
<tr>
<td>$S_D(D) = D$</td>
<td>S2D2</td>
</tr>
<tr>
<td>$S_D(x ≤ a \triangleright y) = S_D(x) ≤ a \triangleright S_D(y)$</td>
<td>S2D3</td>
</tr>
</tbody>
</table>

that only contain basic actions from $A$. We write $p \in p'$, where $p, p' \in T_{\text{finrec}}$, to indicate that $p$ is a subterm of a term $p'' \in T_{\text{finrec}}$ 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.

A thread vector is a sequence of threads. A strategic interleaving operator turns a thread vector of arbitrary length into a single thread. The resulting single thread is sometimes called a multi-thread. Several kinds of strategic interleaving have been elaborated in earlier work, see e.g. [3, 5]. In this section, we only cover one of the simplest interleaving strategies, namely cyclic interleaving with perfect forking.

It is assumed that a fixed but arbitrary thread forking function $\phi : \mathbb{N} \to T_{\text{finrec}}$, where $\mathbb{N} \subseteq \mathbb{N}$, has been given. Moreover, it is assumed that $n(t(n)) \in A$ for all $n \in \text{dom}(\phi)$. The basic action $n(t(n))$ 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 $\{n(t(n)) \mid n \in \text{dom}(\phi)\}$. The strategic interleaving operator for cyclic interleaving with perfect forking is denoted by $∥_I(\_)$.

The axioms for cyclic interleaving with perfect forking are given in Table 4.\footnote{We write $\langle \_ \rangle$ for the empty sequence, $\langle d \rangle$ for the sequence having $d$ as sole element, and $\alpha \triangleright \beta$ for the concatenation of sequences $\alpha$ and $\beta$. We assume that the identities $\alpha \triangleright \langle \_ \rangle = \langle \_ \rangle \triangleright \alpha = \alpha$ hold.}

In CSIf3, the auxiliary deadlock at termination operator $S_D(\_)$ is used. It turns termination into deadlock. Its axioms are given in Table 5. In Table 4, $\alpha$ stands
for an arbitrary member of $A_{\text{tau}} \setminus NT$. In Table 5, $a$ stands for an arbitrary member of $A_{\text{tau}}$.

In [3], 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 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 [5, 4]. 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 $\cdot H \cdot$ to BTA.

A **Maurer machine** is a tuple $H = (M, B, S, I, A, [\cdot])$, where $(M, B, S, I)$ is a Maurer computer and:

- $A \subseteq A \setminus NT$;
- $[\cdot] : A \to (I \times M)$.

The members of $A$ are called the **basic actions** of $H$, and $[\cdot]$ is called the **basic action interpretation function** of $H$.

The apply operators associated with Maurer machines are related to the apply operators introduced in [6]. They allow for threads to transform states of the associated Maurer machine by means of its instructions. Such state transformations produce either a state of the associated Maurer machine or the **undefined state** $\uparrow$. It is assumed that $\uparrow$ is not a state of any Maurer machine. We extend function restriction to $\uparrow$ by stipulating that $\uparrow \restriction M = \uparrow$ for any set $M$.

The first operand of the apply operator $\cdot H \cdot$ associated with Maurer machine $H = (M, B, S, I, A, [\cdot])$ must be a term from $T_{\text{finrec}}(A)$ and its second argument must be a state from $S \cup \{\parallel\}$.

Let $H = (M, B, S, I, A, [\cdot])$ be a Maurer machine, let $p \in T_{\text{finrec}}(A)$, and let $S \in S$. Then $p \cdot H \cdot S$ is the state from $S$ that results if all basic actions performed by thread $p$ are processed by the Maurer machine $H$ from initial state $S$. Moreover, let $(I_a, m_a) = [a]$ for all $a \in A$. Then the processing of a basic action $a$ by $H$ amounts to a state change according to the instruction $I_a$.

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 $H = (M, B, S, I, A, [\cdot])$ be a Maurer machine, and let $(I_a, m_a) = [a]$ for all $a \in A$. Then the apply operator $\cdot H \cdot$ is defined by the equations given in Table 6 (for $a \in A$ and $S \in S$) and the rule given in Table 7 (for $S \in S$). We say that $p \cdot H \cdot S$ is **convergent** if $\exists n \in \mathbb{N} \cdot \pi_n(p)\cdot H \cdot S \neq \uparrow$. If $p \cdot H \cdot S$ is convergent, then the **length of the computation of $p \cdot H \cdot S$**, written $\vert p \cdot H \cdot S \vert$, is the smallest $n \in \mathbb{N}$ such that $\pi_n(p)\cdot H \cdot S \neq \uparrow$. If $p \cdot H \cdot S$ is not convergent, then $\vert p \cdot H \cdot S \vert$ is undefined.

We say that $p \cdot H \cdot S$ is **divergent** if $p \cdot H \cdot S$ is not convergent. Notice that the rule from Table 7 can be read as follows: if $x \cdot H \cdot S$ is divergent, then it equals $\uparrow$. 

7
If $p, S \in S$ then the bijection $m$, Maurer machine. In this section, we make precise how to represent threads in the memory of a Maurer machine.

### Representation of Threads

We introduce some auxiliary notions, which are useful in proofs. Let $H = (M, B, S, I, A, [\_])$ be a Maurer machine, and let $(I_a, m_a) = [a]$ for all $a \in A$. Then the step relation $\vdash_H \subseteq (T_{\text{finrec}}(A) \times S) \times (I_{\text{finrec}}(A) \times S)$ is inductively defined as follows:

- if $p = \tau \circ p'$, then $(p, S) \vdash_H (p', S)$;
- if $I_a(S)(m_a) = T$ and $p = p' \equiv a \equiv p''$, then $(p, S) \vdash_H (p', I_a(S))$;
- if $I_a(S)(m_a) = F$ and $p = p' \equiv a \equiv p''$, then $(p, S) \vdash_H (p'', I_a(S))$.

If $(p, S) \vdash_H (p', S')$, then $p \bullet_H S = p' \bullet_H S'$. Moreover, let $p \in T_{\text{finrec}}(A)$, and let $S \in S$. Then the full path of $p \bullet_H S$ is the unique full path in $\vdash_H$ from $(p, S)$. 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 exists no $(p_{n+1}, S_{n+1}) \in T_{\text{finrec}}(A) \times S$ 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 $\vdash_H$.

If $p \bullet_H S$ is convergent, then its full path is a path of length $\left| p \bullet_H S \right|$ from $(p, S)$ to $(S, S')$, where $S' = p \bullet_H S$. Such a full path is also called a computation.

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

### Table 6. Defining equations for apply operator

<table>
<thead>
<tr>
<th>$x \bullet_H \top$</th>
<th>$\top$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \bullet_H S$</td>
<td>$S$</td>
</tr>
<tr>
<td>$(\tau \circ x) \bullet_H S = x \bullet_H S$</td>
<td></td>
</tr>
<tr>
<td>$(x \equiv y) \bullet_H S = x \bullet_H I_a(S)$</td>
<td>if $I_a(S)(m_a) = T$</td>
</tr>
</tbody>
</table>
| $(x \equiv y) \bullet_H S = y \bullet_H I_a(S)$ | if $I_a(S)(m_a) = F$

### Table 7. Rule for divergence

$$\forall n \in \mathbb{N} \bullet_n(x) \bullet_H S = \top$$

$$x \bullet_H S = \top$$

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 $M_{\text{thr}}$ and a fixed but arbitrary bijection $m_{\text{thr}}: [0, \text{card}(M_{\text{thr}}) - 1] \rightarrow M_{\text{thr}}$ have been given. $M_{\text{thr}}$ is called the thread memory. We write $\text{size}(M_{\text{thr}})$ for $\text{card}(M_{\text{thr}})$. Let $n, n' \in [0, \text{size}(M_{\text{thr}}) - 1]$ be such that $n \leq n'$. Then, we write $M_{\text{thr}}[n]$ for $m_{\text{thr}}(n)$, and $M_{\text{thr}}[n, n']$ for $\{m_{\text{thr}}(k) \mid n \leq k \leq n'\}$. 8
The thread memory is a memory of which the elements can be addressed by means of members of \([0, \text{size}(M_{thb}) - 1]\). We write \(A_{thb}\) for \([0, \text{size}(M_{thb}) - 1]\).

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 \(A_{thb}\) addressing thread memory elements containing representations of other nodes.

Let \(n, n' \in A_{thb}\) be such that \(n \leq n'\). Then, we write \(B_{thb}[n, n']\) for \(\{S, D\} \cup ([n, n'] \times A \times [n, n'])\). We write \(B_{thb}\) for \(B_{thb}[0, \text{size}(M_{thb}) - 1]\). \(B_{thb}\) is called the thread memory base set. We write \(S_{thb}\) for the set of all functions \(S_{thb}: M_{thb} \rightarrow B_{thb}\).

Let \(p \in T_{finrec}\) be a term not containing \(\tau\). Then the nodes of the graph representation of \(p\), written \(\text{Nodes}(p)\), is the smallest subset of \(T_{finrec}\) such that:

- \(p \in \text{Nodes}(p)\);
- if \(p' \leq a \geq p'' \in \text{Nodes}(p)\), then \(p', p'' \in \text{Nodes}(p)\);
- if \(\langle X_0|\{X_0 = t_0, \ldots, X_n = t_n\}\rangle \in \text{Nodes}(p)\), \(\langle t_0|\{X_0 = t_0, \ldots, X_n = t_n\}\rangle \equiv p' \leq a \geq p''\), then \(p', p'' \in \text{Nodes}(p)\).

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

It is assumed that for all \(p \in T_{finrec}\), a fixed but arbitrary bijection \(\text{node}_p: [0, \text{size}(p) - 1] \rightarrow \text{Nodes}(p)\) with \(\text{node}_p(0) = p\) has been given.

Let \(p \in T_{finrec}\) be a term not containing \(\tau\), with \(\text{size}(p) \leq \text{size}(M_{thb})\). Then the stored graph representation of \(p\), written \(s_{thb}(p)\), is the unique function \(s_{thb}: M_{thb}[0, \text{size}(p) - 1] \rightarrow B_{thb}[0, \text{size}(p) - 1]\) such that for all \(n \in [0, \text{size}(p) - 1]\), \(s_{thb}(M_{thb}[n]) = \text{nrepr}_p(\text{node}_p(n))\), where \(\text{nrepr}_p: \text{Nodes}(p) \rightarrow B_{thb}[0, \text{size}(p) - 1]\) is defined as follows:

\[
\text{nrepr}_p(S) = S,
\text{nrepr}_p(D) = D,
\text{nrepr}_p(p' \leq a \geq p'') = (\text{node}_p^{-1}(p'), a, \text{node}_p^{-1}(p'')),
\text{nrepr}_p(\langle X_0|\{X_0 = t_0, \ldots, X_n = t_n\}\rangle) = \text{nrepr}_p(\langle t_0|\{X_0 = t_0, \ldots, X_n = t_n\}\rangle).
\]

We call \(s_{thb}(p)\) a stored thread.

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

7 Turing Machines

Turing machines were proposed almost 70 years ago by Turing in [13]. 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. [11, 9]): 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.
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 □. 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, □\} for the set of tape symbols. We write \(B_{\text{tape}}\) for the set \(\{0, 1, □\}\). 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 \(D_{\text{hd}}\) for the set \(\{L, R\}\).

A simple Turing machine \(T\) consists of the following components:

- a finite set \(Q\);
- a function \(\delta : Q \times B_{\text{tape}} \rightarrow Q \times (B_{\text{tape}} \cup D_{\text{hd}} \cup \{H\})\);
- an element \(q^0 \in Q\).

The members of \(Q\) are called the states, \(\delta\) 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 \(\tau : \mathbb{N} \rightarrow B_{\text{tape}}\) for which there exists an \(n \in \mathbb{N}\) such that for all \(m \in \mathbb{N}\) with \(m \geq n\), \(\tau(m) = □\). We write \(C_{\text{tape}}\) 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 \(q \in Q\), \(τ \in C_{\text{tape}}\) and \(i \in \mathbb{N}\). If \(q = q^0\), then \((q, τ, i)\) is an initial configuration of \(T\). If \(δ(q, τ(i)) = (q', H)\) 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 \(s' \in B_{\text{tape}} \cup D_{\text{hd}}\) such that \(s' \neq L\) if \(i = 0\):

\[
δ(q, τ(i)) = (q', s') ,
\]

for all \(j \in \mathbb{N}\):

\[
\begin{align*}
τ'(j) &= s' & \text{if } i = j \land s' \in B_{\text{tape}} , \\
τ'(j) &= τ(j) & \text{if } i = j \land s' \in D_{\text{hd}} , \\
τ'(j) &= τ(j) & \text{if } i \neq j , \\
\end{align*}
\]

and

\[
\begin{align*}
i' &= i & \text{if } s' \in B_{\text{tape}} , \\
i' &= i - 1 & \text{if } s' = L , \\
i' &= i + 1 & \text{if } s' = R . \\
\end{align*}
\]

Let \(T = (Q, δ, q^0)\) be a simple Turing machine. Then the step relation \(\triangleright_T\) \(\subseteq (Q \times C_{\text{tape}} \times \mathbb{N}) \times (Q \times C_{\text{tape}} \times \mathbb{N})\) is defined by \((q, τ, i) \triangleright_T (q', τ', i')\)
iff \((q', \tau', i')\) is next to \((q, \tau, i)\) in \(T\). A \textit{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 \((q_0, \tau_0, i_0)\) is an initial configuration of \(T\) and \((q_n, \tau_n, i_n)\) is a terminal configuration of \(T\).\footnote{We interpret the usual remark “no left move is permitted when the read-write head is at the [left] boundary” (see e.g. [11]) 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”}

In the case of a simple Turing machine, the set of tape symbols is invariably \(B_{\text{tape}}\), 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. [9]). 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 be simulated by a simple Turing machine (for details, see e.g. [8, 7, 11, 9]). 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.

8 Simulation of Turing Machines

In this section, we show a straightforward 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 \(M_{\text{tape}}\) and a fixed but arbitrary bijection \(m_{\text{tape}} : \mathbb{N} \rightarrow M_{\text{tape}}\) have been given. \(M_{\text{tape}}\) is called a \textit{tape memory}. Let \(n \in \mathbb{N}\). Then we write \(M_{\text{tape}}[n]\) for \(m_{\text{tape}}(n)\).

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 \(\Box\). We write \(S_{\text{tape}}\) for the set of all functions \(S_{\text{tape}} : M_{\text{tape}} \rightarrow B_{\text{tape}}\) for which there exists an \(n \in \mathbb{N}\) such that for all \(m \in \mathbb{N}\) with \(m \geq n\), \(S_{\text{tape}}(M_{\text{tape}}[m]) = \Box\).

The Maurer machines \(TM, TM'\) and \(TM''\) defined in this paper would not be Maurer machines if \(S_{\text{tape}}\) was simply the set of all functions \(S_{\text{tape}} : M_{\text{tape}} \rightarrow B_{\text{tape}}\), for Maurer machines may not have states that differ in the contents of infinitely many memory elements.

The memory of the Maurer machine \(TM\) used to simulate Turing machines consists of a \textit{tape memory} (\(M_{\text{tape}}\)), a \textit{head position register} (\(\text{head}\)) and a \textit{reply register} (\(rr\)). Its instruction set consists of two \textit{test} instructions (\(I_{\text{test.}0}, I_{\text{test.}1}\)), two \textit{write} instructions (\(I_{\text{write.}0}, I_{\text{write.}1}\)) and two \textit{move head} instructions (\(I_{\text{mover}}, I_{\text{movel}}\)). Each instruction is associated with a basic action. The set of basic actions of \(TM\) consists of those basic actions.

It is assumed that \(\text{test.} : s, \text{write.} : s \in \mathcal{A}\), for all \(s \in B_{\text{tape}}\), and \(\text{mover}, \text{movel} \in \mathcal{A}\).
\(TM\) is the Maurer machine \((M, B, S, I, A, [\_])\) such that

\[
\begin{align*}
M &= M_{\text{tape}} \cup \{\text{head, rr}\}, \\
B &= B_{\text{tape}} \cup \mathbb{N} \cup \mathbb{B}, \\
S &= \{S : M \rightarrow B \mid S \models M_{\text{tape}} \land S(\text{head}) \in \mathbb{N} \land S(\text{rr}) \in \mathbb{B}\}, \\
I &= \{I_{\text{test}, s}, I_{\text{write}, s} \mid s \in B_{\text{tape}}\} \cup \{I_{\text{mover}, I_{\text{move}}}\}, \\
A &= \{\text{test}: s, \text{write}: s \mid s \in B_{\text{tape}}\} \cup \{\text{mover}, \text{move}\}, \\
[a] &= (I_{a}, \text{rr}) \quad \text{for all } a \in A.
\end{align*}
\]

Here, for each \(s \in B_{\text{tape}}, I_{\text{test}, s}\) is the unique function from \(S\) to \(S\) such that for all \(S \in S\):

\[
\begin{align*}
I_{\text{test}, s}(S) \upharpoonright M_{\text{tape}} &= S \upharpoonright M_{\text{tape}}, \\
I_{\text{test}, s}(S)(\text{head}) &= S(\text{head}), \\
I_{\text{test}, s}(S)(\text{rr}) &= T \quad \text{if } S(M_{\text{tape}}[S(\text{head})]) = s, \\
I_{\text{test}, s}(S)(\text{rr}) &= F \quad \text{if } S(M_{\text{tape}}[S(\text{head})]) \neq s;
\end{align*}
\]

for each \(s \in B_{\text{tape}}, I_{\text{write}, s}\) is the unique function from \(S\) to \(S\) such that for all \(S \in S\) and \(n \in \mathbb{N}\):

\[
\begin{align*}
I_{\text{write}, s}(S)(M_{\text{tape}}[S(\text{head})]) &= s, \\
I_{\text{write}, s}(S)(M_{\text{tape}}[n]) &= S(M_{\text{tape}}[n]) \quad \text{if } S(\text{head}) \neq n, \\
I_{\text{write}, s}(S)(\text{head}) &= S(\text{head}), \\
I_{\text{write}, s}(S)(\text{rr}) &= 1;
\end{align*}
\]

\(I_{\text{mover}}\) is the unique function from \(S\) to \(S\) such that for all \(S \in S\):

\[
\begin{align*}
I_{\text{mover}}(S) \upharpoonright M_{\text{tape}} &= S \upharpoonright M_{\text{tape}}, \\
I_{\text{mover}}(S)(\text{head}) &= S(\text{head}) + 1, \\
I_{\text{mover}}(S)(\text{rr}) &= 1;
\end{align*}
\]

and, \(I_{\text{move}}\) is the unique function from \(S\) to \(S\) such that for all \(S \in S\):

\[
\begin{align*}
I_{\text{move}}(S) \upharpoonright M_{\text{tape}} &= S \upharpoonright M_{\text{tape}}, \\
I_{\text{move}}(S)(\text{head}) &= S(\text{head}) - 1 \quad \text{if } S(\text{head}) > 0, \\
I_{\text{move}}(S)(\text{head}) &= 0 \quad \text{if } S(\text{head}) = 0, \\
I_{\text{move}}(S)(\text{rr}) &= 0 \quad \text{if } S(\text{head}) > 0, \\
I_{\text{move}}(S)(\text{rr}) &= F \quad \text{if } S(\text{head}) = 0.
\end{align*}
\]

We write \(S_{TM}\) and \(A_{TM}\) for the set of states of \(TM\) and the set of basic actions of \(TM\), respectively.

A Turing thread is a constant \(\langle X_0|\{X_0 = t_0, \ldots, X_n = t_n\}\rangle \in T_{\text{finrec}}\), where \(t_0, \ldots, t_n\) are terms of the form \(t \leq \text{test}: 0 \geq (t' \leq \text{test}: 1 \geq t'')\) with \(t, t'\) and \(t''\) of the form \(\text{write}: s \circ X, \text{mover}: X, X \leq \text{move} \geq D\) or \(S\) \((s \in B_{\text{tape}}, X \in \{X_0, \ldots, X_n\})\).

Each Turing machine can be simulated on the Maurer machine \(TM\) by means of a Turing thread \(p \in T_{\text{finrec}}(A_{TM})\).
Theorem 1. Let $T = (Q, \delta, q_0)$ be a Turing machine. Then there exists a Turing thread $p \in \mathcal{T}_{\text{finrec}}(A_{TM})$ such that for all computations $c$ of $T$, there exists an $S \in \mathcal{S}_{TM}$ such that the computation of $p \cdot TM \cdot S$ simulates $c$.

Proof. Suppose that $Q = \{q_0, \ldots, q_n\}$. Let $E$ be the guarded recursive specification \( \{ X_i = t_{i0} \triangleq \text{test:0} \triangleright (t_{i1} \triangleq \text{test:1} \triangleright t_{i\square}) \mid 0 \leq i \leq n \} \), where

\[
\begin{align*}
\phi_{s} & : X_j = \text{write:s'} \triangleright X_j \quad \text{if } \delta(q_i, s) = (q_j, s') \land s' \in \mathcal{B}_{\text{tape}}, \\
\phi_{s} & : X_j = \text{mover} \circ X_j \quad \text{if } \delta(q_i, s) = (q_j, R), \\
\phi_{s} & : X_j \triangleq \text{movel} \triangleright D \quad \text{if } \delta(q_i, s) = (q_j, L), \\
\phi_{s} & : S \quad \text{if } \exists q \in Q \cdot \delta(q_i, s) = (q, H).
\end{align*}
\]

Define $\phi : Q \rightarrow \mathcal{T}_{\text{finrec}}(A_{TM})$ by $\phi(q_i) = \langle X_i | E \rangle$ $(0 \leq i \leq n)$. Clearly, $\phi(q_i)$ is a Turing thread. Define $\phi' : \mathcal{C}_{\text{tape}} \times \mathbb{N} \rightarrow S_{TM}$ by $\phi'(q, i)$ is the unique state $S \in S_{TM}$ such that $S(M_{\text{tape}}[j]) = \tau(j)$ for all $j \in \mathbb{N}$, $S(\text{head}) = i$ and $S(\text{rr}) = T$. Combine $\phi$ and $\phi'$ to $\phi'' : Q \times \mathcal{C}_{\text{tape}} \times \mathbb{N} \rightarrow \mathcal{T}_{\text{finrec}}(A_{TM}) \times S_{TM}$ defined by $\phi''(q, \tau, i) = (\phi(q), \phi'(q, i))$. Then we have $(q, \tau, i) \vdash T (q', \tau', i')$ iff $\phi''(q, \tau, i) \vdash T \phi''(q', \tau', i')$. This is easily proved by distinction between the following cases: $\delta(q, \tau(i)) \in Q \times \mathcal{B}_{\text{tape}}, \delta(q, \tau(i)) \in Q \times \{ R \}, \delta(q, \tau(i)) \in Q \times \{ L \}, \delta(q, \tau(i)) \in Q \times \{ H \}$. 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 $\phi(q_0) \cdot TM \cdot \phi'(\tau_0, i_0)$ simulates $c$. \qed

Looking at the instructions used in the simulation of Turing machines on the Maurer machine $TM$, we observe that the test instructions $I_{\text{test:s}}$ have an infinite input region and a finite output region and that the write instructions $I_{\text{write:s}}$ 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 that, starting from head position 0, 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 for this Turing machine because the first cell holding $\Box$ may occur anywhere on the tape and the Turing machine has 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 instructions with a finite input region and a finite output region.

9 Using Instructions with Finite Input & Output Regions

In order to simulate Turing machines using test and write instructions with a finite input region and a finite output region, we have to adapt the Maurer machine $TM$. 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 $TM$ by removing the head position register head from the memory and the move head instructions $I_{\text{movel}}$ and $I_{\text{mover}}$ from the instruction set. In addition, we replace each test instruction $I_{\text{test:s}}$ by a test
instruction \( I_{\text{test},s:n} \) for each \( n \in \mathbb{N} \), and each write instruction \( I_{\text{write},s:n} \) by a write instruction \( I_{\text{write},s:n} \) for each \( n \in \mathbb{N} \). Each instruction is associated with a basic action. We replace the basic actions of \( TM \) by those basic actions.

It is assumed that \( \text{test},s:n, \text{write},s:n \in A \) for all \( s \in B_{\text{tape}} \) and \( n \in \mathbb{N} \). \( TM' \) is the Maurer machine \((M, B, S, I, A, [\cdot])\) such that

\[
\begin{align*}
M &= M_{\text{tape}} \cup \{rr\}, \\
B &= B_{\text{tape}} \cup B, \\
S &= \{S : M \to B \mid S(M_{\text{tape}}) \in S_{\text{tape}} \land S(rr) \in B\}, \\
I &= \{I_{\text{test},s:n}, I_{\text{write},s:n} \mid s \in B_{\text{tape}} \land n \in \mathbb{N}\}, \\
A &= \{\text{test},s:n, \text{write},s:n \mid s \in B_{\text{tape}} \land n \in \mathbb{N}\},
\end{align*}
\]

Here, for each \( s \in B_{\text{tape}} \) and \( n \in \mathbb{N} \), \( I_{\text{test},s:n} \) is the unique function from \( S \) to \( S \) such that for all \( S \in S \):

\[
\begin{align*}
I_{\text{test},s:n}(S)(M_{\text{tape}}) &= S(M_{\text{tape}}), \\
I_{\text{test},s:n}(S)(rr) &= T \quad \text{if } S(M_{\text{tape}}) = s, \\
I_{\text{test},s:n}(S)(rr) &= F \quad \text{if } S(M_{\text{tape}}) \neq s;
\end{align*}
\]

for each \( s \in B_{\text{tape}} \) and \( n \in \mathbb{N} \), \( I_{\text{write},s:n} \) is the unique function from \( S \) to \( S \) such that for all \( S \in S \) and \( m \in \mathbb{N} \):

\[
\begin{align*}
I_{\text{write},s:n}(S)(M_{\text{tape}}[n]) &= s, \\
I_{\text{write},s:n}(S)(M_{\text{tape}}[m]) &= S(M_{\text{tape}}[m]) \quad \text{if } n \neq m, \\
I_{\text{write},s:n}(S)(rr) &= T.
\end{align*}
\]

We write \( S_{TM'} \) and \( A_{TM'} \) for the set of states of \( TM' \) and the set of basic actions of \( TM' \), respectively.

Let \( (X_0|E), \) 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 subterms of a term \( t \in T_{\text{finrec}} \) for which \( t_0 = t \) is derivable from \( E \). Then the relation \( \mathcal{H}_{X_i} \subseteq T_0 \times (\mathbb{N} \cup \{-1\}) \) is inductively defined as follows:

- if \( X_i \in T_0 \), then \( \mathcal{H}_{X_i}(X_i, 0) \);
- if \( \mathcal{H}_{X_i}(t, l) \), \( t \leq \text{test},s \geq t' \in T_0 \) and \( l \geq 0 \), then \( \mathcal{H}_{X_i}(t \leq \text{test},s \geq t', l) \);
- if \( \mathcal{H}_{X_i}(t, l) \), \( t' \leq \text{test},s \geq t \in T_0 \) and \( l \geq 0 \), then \( \mathcal{H}_{X_i}(t' \leq \text{test},s \geq t, l) \);
- if \( \mathcal{H}_{X_i}(t, l) \), \( \text{write},s \circ t \in T_0 \) and \( l \geq 0 \), then \( \mathcal{H}_{X_i}(\text{write},s \circ t, l) \);
- if \( \mathcal{H}_{X_i}(t, l) \), \( \text{mover} \circ t \in T_0 \) and \( l \geq 0 \), then \( \mathcal{H}_{X_i}(\text{mover} \circ t, l + 1) \);
- if \( \mathcal{H}_{X_i}(t, l) \), \( t \leq \text{movel} \geq D \in T_0 \) and \( l \geq 0 \), then \( \mathcal{H}_{X_i}(t \leq \text{movel} \geq D, l - 1) \).

We write \( X_0 \stackrel{k}{\rightarrow}_E X_i \) to indicate that there exist a \( t \in T_{\text{finrec}} \) such that \( t_0 = t \) is derivable from \( E \) and \( \mathcal{H}_{X_i}(t,k) \) holds. The recursive specification \( \psi(E) \) is inductively defined as follows:

- if \( X_0 \stackrel{k}{\rightarrow}_E X_i \), then \( X_{ik} = t_{ik} \in \psi(E) \),

where \( t_{ik} \) is obtained from \( t_i \) by applying the following replacement rules:
• test: is replaced by test:\;s:k;
• write: is replaced by write: s:k \circ X_j;
• mover \circ X_j is replaced by mover: s:k \circ X_j;
• if \( k \neq 0 \), then \( X_j \preceq mover: D \) is replaced by \( X_j \preceq D \), where \( l = k + 1 \);
• if \( k = 0 \), then \( X_j \preceq mover: D \) is replaced by \( D \).

We write \( \psi(\langle X_0 \rangle E) \) for \( \langle X_0 \rangle \psi(E) \).

The variables of a Turing thread \( p \) correspond to the states of a Turing machine. If the head position is made part of the instructions, 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 \( TM \) from some state of \( TM \) has the same effect as applying \( \psi(p) \) to Maurer machine \( TM' \) from the corresponding state of \( TM' \).

**Theorem 2.** Let \( p \) be a Turing thread, and let \( S_0 \in S_{TM} \) and \( S'_0 \in S'_{TM} \) be such that \( S_0 \upharpoonright (M_{tape} \cup \{rr\}) = S'_0 \) and \( S_0(\text{head}) = 0 \). Then \( (p \bullet_{TM} S_0) \upharpoonright (M_{tape} \cup \{rr\}) = \psi(p) \bullet_{TM'} S'_0 \).

**Proof.** It is easy to see that for all \( S \in S_{TM} \):

\[
\begin{align*}
I_{\text{test}:s}(S) \upharpoonright (M_{tape} \cup \{rr\}) &= I_{\text{test}:s:n}(S \upharpoonright (M_{tape} \cup \{rr\})) , \\
I_{\text{write}:s}(S) \upharpoonright (M_{tape} \cup \{rr\}) &= I_{\text{write}:s:n}(S \upharpoonright (M_{tape} \cup \{rr\})) , \\
I_{\text{mover}}(S) \upharpoonright M_{tape} &= S \upharpoonright M_{tape} , \\
I_{\text{movel}}(S) \upharpoonright M_{tape} &= S \upharpoonright M_{tape} ,
\end{align*}
\]

where \( n = S(\text{head}) \).

Let \( (p_n, S_n) \) be the \( n+1 \)-th element in the full path of \( p \bullet_{TM} S_0 \) of which the first component equals \( S, D \) or \( q \preceq \text{test}:0 \preceq r \) for some \( q, r \in T_{\text{finrec}} \), and let \( (p'_n, S'_n) \) be the \( n+1 \)-th element in the full path of \( \psi(p) \bullet_{TM'} (S_0 \upharpoonright (M_{tape} \cup \{rr\})) \) of which the first component equals \( S, D \) or \( q' \preceq \text{test}:0 \preceq r' \) for some \( k, r' \in \mathbb{N} \) and \( q', r' \in T_{\text{finrec}} \). Then, using the above equations, it is straightforward to prove by induction on \( n \) that:

\[
- \quad p_n = q \preceq \text{test}:0 \preceq r \text{ and } S_n(\text{head}) = k \text{ if } p'_n = q' \preceq \text{test}:0 \preceq r' \text{ with } q' \text{ and } r' \text{ obtained from } q \text{ and } r \text{ by applying the replacement rules given in the definition of } \psi \text{ above;} \\
- \quad S_n \upharpoonright (M_{tape} \cup \{rr\}) = S'_n.
\]

(for \( n < \| p \bullet_{TM} S_0 \| \) if \( p \bullet_{TM} S_0 \) is convergent). From this, the theorem follows immediately. \( \square \)

Theorem 2 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 \( TM' \) 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.
Consider again the Turing machine that, starting from head position 0, overwrites cells that hold 0 with 1 and cells that hold 1 with 0 and halts when the first cell holding □ is reached. The Turing thread that corresponds to the finite-state control of this Turing machine is the constant \( \langle X_0 | E \rangle \), where

\[
E = \{ X_0 = \text{write:}1 \circ X_1 \triangleleft \text{test:}0 \triangleright (\text{write:}0 \circ X_1 \triangleleft \text{test:}1 \triangleright S), \\
X_1 = \text{mover} \circ X_0 \triangleleft \text{test:}0 \triangleright (\text{mover} \circ X_0 \triangleleft \text{test:}1 \triangleright \text{mover} \circ X_0) \}.
\]

Clearly, \( X_0 \xrightarrow{k} E X_0 \) for each \( k \in \mathbb{N} \). Consequently, in this case \( \psi(E) \) is an infinite-state thread. We will show in Section 10 that, when simulating Turing machines on a Maurer machine using test and write instructions with a finite input region and a finite output region, we can get around infinite-state threads in the case of convergent computations.

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 instruction implicitly executed by a Turing machine on a test step has an infinite input region and a finite output region;
– the instruction implicitly executed by a Turing machine on a write step has a finite input region and an infinite output region;
– these instructions can be replaced by instructions 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 instructions with a finite input region and a finite output region that gets around infinite-state threads in the case of convergent computations. 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 instructions;
– 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 if the computation concerned is convergent. Although it is not necessary to get around infinite-state threads, the head position register \( \text{head} \) will be replaced by a countably infinite memory.

It is assumed that a fixed but arbitrary countably infinite set \( M_\text{hd} \) and a fixed but arbitrary bijection \( m_\text{hd} : \mathbb{N} \rightarrow M_\text{hd} \) have been given. \( M_\text{hd} \) is called a head position memory. Let \( n \in \mathbb{N} \). Then we write \( M_\text{hd}[n] \) for \( m_\text{hd}(n) \). \( M_\text{hd} \) is an infinite memory of which the elements can be addressed by means of members of...
The elements of $M_{hd}$ contain $T$ or $F$. We write $S_{hd}$ for the set of all functions $S_{hd} : M_{hd} \to \mathbb{B}$ for which there exists an $n \in \mathbb{N}$ such that $S_{hd}[M_{hd}[n]] = T$ and for all $m \in \mathbb{N}$ with $m \neq n$. $S_{hd}[M_{hd}[m]] = F$. $M_{hd}$ will be used in such a way that the head position is always the unique $n$ for which $M_{hd}[n]$ contains $T$. The memory of the simulating Maurer machine remains countably infinite if $\text{head}$ is replaced by $M_{hd}$. The replacement achieves that, for each memory element, all possible contents belong to a finite set.

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

We adapt the Maurer machine $TM$ by extending the memory with a thread memory $(M_{thr})$, a thread location register $(\text{tlr})$ and a basic action register $(\text{bar})$, and the instruction set with a halt instruction $(\text{halt})$, a fetch instruction $(\text{fetch})$, an execute stored basic action instruction $(\text{exsba}_{n})$ for each $n \in \mathbb{N}$, an test execution mode instruction $(\text{testem})$, and a test head position instruction $(\text{testhp}_{n})$ for each $n \in \mathbb{N}$. Each of these extra instructions is associated with a basic action. We replace the basic actions of $TM$ by those basic actions. In addition, we replace the head position register $\text{head}$ by a head position memory $M_{hd}$, each test instruction $\text{test}_{s,n}$ by a test instruction $\text{test}_{s,n}$ for each $n \in \mathbb{N}$, and each write instruction $\text{write}_{s,n}$ by a write instruction $\text{write}_{s,n}$ for each $n \in \mathbb{N}$.

The thread memory $M_{thr}$, 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 $M_{thr}$, in the basic action register $\text{bar}$ and then executing the basic action in $\text{bar}$. The thread location register $\text{tlr}$ 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, $\text{tlr}$ 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 instruction $\text{exsba}_{n}$ allows for making the head position part of the instruction that corresponds to the basic action in $\text{bar}$. The instruction $\text{testhp}_{n}$ is meant for testing whether the head position is $n$. The instruction $\text{testem}$ allows for testing whether the execution of the stored Turing thread has not yet come to an end.

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

$TM''$ is the Maurer machine $(M, B, S, I, A, [\_])$ such that

$$M = M_{\text{tape}} \cup M_{hd} \cup M_{thr} \cup \{\text{tlr}, \text{bar}, \text{rr}\},$$

$$B = B_{\text{tape}} \cup B \cup B_{\text{thr}} \cup A_{\text{thr}} \cup \{-1\} \cup A_{TM},$$

$$S = \{S : M \to B \mid S|_{M_{\text{tape}}} \in S_{\text{tape}} \land S|_{M_{hd}} \in S_{hd} \land S|_{M_{thr}} \in S_{thr} \land S(\text{tlr}) \in A_{\text{thr}} \cup \{-1\} \land S(\text{bar}) \in A_{TM} \land S(\text{rr}) \in \mathbb{B}\} ,$$
\[ \mathcal{I} = \{ I_{\text{test}:s:n}, I_{\text{write}:s:n} \mid s \in B_{\text{tape}} \land n \in \mathbb{N} \} \cup \{ I_{\text{move}}, I_{\text{novel}} \} \]

\[ \cup \{ I_{\text{testem}}, I_{\text{halt}}, I_{\text{fetch}} \} \cup \{ I_{\text{testhp}, n}, I_{\text{exsba:n}} \mid n \in \mathbb{N} \} , \]

\[ A = \{ \text{testem, halt, fetch} \} \cup \{ \text{testhp:n, exsba:n} \mid n \in \mathbb{N} \} , \]

\[ [a] = (I_a, r) \quad \text{for all } a \in A . \]

Here, \( I_{\text{testem}} \) is the unique function from \( S \) to \( S \) such that for all \( S \in S \):

\[ I_{\text{testem}}(S) \upharpoonright M_{\text{tape}} = S \upharpoonright M_{\text{tape}} , \]

\[ I_{\text{testem}}(S) \upharpoonright M_{\text{hd}} = S \upharpoonright M_{\text{hd}} , \]

\[ I_{\text{testem}}(S) \upharpoonright M_{\text{thr}} = S \upharpoonright M_{\text{thr}} , \]

\[ I_{\text{testem}}(S)(\text{thr}) = S(\text{thr}) , \]

\[ I_{\text{testem}}(S)(\text{bar}) = S(\text{bar}) , \]

\[ I_{\text{testem}}(S)(\text{rr}) = T \quad \text{if } S(M_{\text{thr}}[S(\text{thr})]) \in \{ S, D \} , \]

\[ I_{\text{testem}}(S)(\text{rr}) = F \quad \text{if } S(M_{\text{thr}}[S(\text{thr})]) \notin \{ S, D \} ; \]

\( I_{\text{halt}} \) is the unique function from \( S \) to \( S \) such that for all \( S \in S \):

\[ I_{\text{halt}}(S) \upharpoonright M_{\text{tape}} = S \upharpoonright M_{\text{tape}} , \]

\[ I_{\text{halt}}(S) \upharpoonright M_{\text{hd}} = S \upharpoonright M_{\text{hd}} , \]

\[ I_{\text{halt}}(S) \upharpoonright M_{\text{thr}} = S \upharpoonright M_{\text{thr}} , \]

\[ I_{\text{halt}}(S)(\text{thr}) = S(\text{thr}) , \]

\[ I_{\text{halt}}(S)(\text{bar}) = S(\text{bar}) , \]

\[ I_{\text{halt}}(S)(\text{rr}) = T \quad \text{if } S(M_{\text{thr}}[S(\text{thr})]) = S , \]

\[ I_{\text{halt}}(S)(\text{rr}) = F \quad \text{if } S(M_{\text{thr}}[S(\text{thr})]) \neq S ; \]

\( I_{\text{fetch}} \) is the unique function from \( S \) to \( S \) such that for all \( S \in S \):

\[ I_{\text{fetch}}(S) \upharpoonright M_{\text{tape}} = S \upharpoonright M_{\text{tape}} , \]

\[ I_{\text{fetch}}(S) \upharpoonright M_{\text{hd}} = S \upharpoonright M_{\text{hd}} , \]

\[ I_{\text{fetch}}(S) \upharpoonright M_{\text{thr}} = S \upharpoonright M_{\text{thr}} , \]

\[ I_{\text{fetch}}(S)(\text{thr}) = \text{ntl}(S, r) , \]

\[ I_{\text{fetch}}(S)(\text{bar}) = \pi_2(S(M_{\text{thr}}[\text{ntl}(S, r)])) \quad \text{if } S(M_{\text{thr}}[\text{ntl}(S, r)]) \notin \{ S, D \} , \]

\[ I_{\text{fetch}}(S)(\text{bar}) = S(\text{bar}) \quad \text{if } S(M_{\text{thr}}[\text{ntl}(S, r)]) \in \{ S, D \} , \]

\[ I_{\text{fetch}}(S)(\text{rr}) = T \quad \text{if } S(M_{\text{thr}}[\text{ntl}(S, r)]) \notin \{ S, D \} , \]

\[ I_{\text{fetch}}(S)(\text{rr}) = F \quad \text{if } S(M_{\text{thr}}[\text{ntl}(S, r)]) \in \{ S, D \} , \]

where \( r = S(\text{rr}) \) and \( \text{ntl} : S \times B \rightarrow A_{\text{thr}} \) is defined as follows:

\[ \text{ntl}(S, T) = \pi_1(S(M_{\text{thr}}[S(\text{thr})])) \quad \text{if } S(\text{thr}) \in A_{\text{thr}} \land S(M_{\text{thr}}[S(\text{thr})]) \notin \{ S, D \} , \]

\[ \text{ntl}(S, F) = \pi_3(S(M_{\text{thr}}[S(\text{thr})])) \quad \text{if } S(\text{thr}) \in A_{\text{thr}} \land S(M_{\text{thr}}[S(\text{thr})]) \notin \{ S, D \} , \]

\[ \text{ntl}(S, r') = S(\text{thr}) \quad \text{if } S(\text{thr}) \in A_{\text{thr}} \land S(M_{\text{thr}}[S(\text{thr})]) \in \{ S, D \} , \]

\[ \text{ntl}(S, r') = 0 \quad \text{if } S(\text{thr}) \notin A_{\text{thr}} ; \]

18
for each $n \in \mathbb{N}$, $I_{\text{test} : s,n}$ is the unique function from $S$ to $S$ such that for all $S \in S$:

\[
I_{\text{test} : s,n}(S) \downharpoonright M_{\text{tape}} = S \downharpoonright M_{\text{tape}} ,
I_{\text{test} : s,n}(S) \downharpoonright M_{\text{hd}} = S \downharpoonright M_{\text{hd}} ,
I_{\text{test} : s,n}(S) \downharpoonright M_{\text{thr}} = S \downharpoonright M_{\text{thr}} ,
I_{\text{test} : s,n}(S)(\text{thr}) = S(\text{thr}) ,
I_{\text{test} : s,n}(S)(\text{bar}) = S(\text{bar}) ,
I_{\text{test} : s,n}(S)(\text{rr}) = S(M_{\text{hd}}[n]) ;
\]

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

\[
I_{\text{exsba} : n}(S) = \text{tmi}(S(\text{bar}), n)(S) ,
\]

where $\text{tmi} : A_{TM} \times \mathbb{N} \rightarrow I$ is defined as follows:

\[
\text{tmi}(\text{test} : s, n) = I_{\text{test} : s,n} ,
\text{tmi}(\text{write} : s, n) = I_{\text{write} : s,n} ,
\text{tmi}(\text{mover}, n) = I_{\text{mover}} ,
\text{tmi}(\text{movel}, n) = I_{\text{movel}} ;
\]

for each $s \in B_{\text{tape}}$ and $n \in \mathbb{N}$, $I_{\text{test} : s,n}$ is the unique function from $S$ to $S$ such that for all $S \in S$:

\[
I_{\text{test} : s,n}(S) \downharpoonright M_{\text{tape}} = S \downharpoonright M_{\text{tape}} ,
I_{\text{test} : s,n}(S) \downharpoonright M_{\text{hd}} = S \downharpoonright M_{\text{hd}} ,
I_{\text{test} : s,n}(S) \downharpoonright M_{\text{thr}} = S \downharpoonright M_{\text{thr}} ,
I_{\text{test} : s,n}(S)(\text{thr}) = S(\text{thr}) ,
I_{\text{test} : s,n}(S)(\text{bar}) = S(\text{bar}) ,
I_{\text{test} : s,n}(S)(\text{rr}) = T \quad \text{if } S(M_{\text{tape}}[n]) = s ,
I_{\text{test} : s,n}(S)(\text{rr}) = F \quad \text{if } S(M_{\text{tape}}[n]) \neq s ;
\]

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

\[
I_{\text{write} : s,n}(S)(M_{\text{tape}}[n]) = s ,
I_{\text{write} : s,n}(S)(M_{\text{tape}}[m]) = S(M_{\text{tape}}[m]) \quad \text{if } n \neq m ,
I_{\text{write} : s,n}(S) \downharpoonright M_{\text{hd}} = S \downharpoonright M_{\text{hd}} ,
I_{\text{write} : s,n}(S) \downharpoonright M_{\text{thr}} = S \downharpoonright M_{\text{thr}} ,
I_{\text{write} : s,n}(S)(\text{thr}) = S(\text{thr}) ,
I_{\text{write} : s,n}(S)(\text{bar}) = S(\text{bar}) ,
I_{\text{write} : s,n}(S)(\text{rr}) = T ;
\]
$I_{\text{mover}}$ is the unique function from $S$ to $S$ such that for all $S \in S$:

\[
\begin{align*}
I_{\text{mover}}(S) \upharpoonright M_{\text{tape}} &= S \upharpoonright M_{\text{tape}}, \\
I_{\text{mover}}(S) \upharpoonright M_{\text{hd}} &= \text{shiftr}(S \upharpoonright M_{\text{hd}}), \\
I_{\text{mover}}(S) \upharpoonright M_{\text{thr}} &= S \upharpoonright M_{\text{thr}}, \\
I_{\text{mover}}(S)(\text{thr}) &= S(\text{thr}), \\
I_{\text{mover}}(S)(\text{bar}) &= S(\text{bar}), \\
I_{\text{mover}}(S)(\text{rl}) &= T,
\end{align*}
\]

where $\text{shiftr} : S_{\text{hd}} \to S_{\text{hd}}$ is defined as follows ($n \in \mathbb{N}$):

\[
\begin{align*}
\text{shiftr}(S)(M_{\text{hd}}[0]) &= F, \\
\text{shiftr}(S)(M_{\text{hd}}[n + 1]) &= S(M_{\text{hd}}[n]);
\end{align*}
\]

and, $I_{\text{mover}}$ is the unique function from $S$ to $S$ such that for all $S \in S$:

\[
\begin{align*}
I_{\text{mover}}(S) \upharpoonright M_{\text{tape}} &= S \upharpoonright M_{\text{tape}}, \\
I_{\text{mover}}(S) \upharpoonright M_{\text{hd}} &= \text{shiftr}(S \upharpoonright M_{\text{hd}}), \\
I_{\text{mover}}(S) \upharpoonright M_{\text{thr}} &= S \upharpoonright M_{\text{thr}}, \\
I_{\text{mover}}(S)(\text{tlr}) &= S(\text{tlr}), \\
I_{\text{mover}}(S)(\text{bar}) &= S(\text{bar}), \\
I_{\text{mover}}(S)(\text{rl}) &= \neg S(M_{\text{hd}}[0]),
\end{align*}
\]

where $\text{shiftr} : S_{\text{hd}} \to S_{\text{hd}}$ is defined as follows ($n \in \mathbb{N}$):

\[
\begin{align*}
\text{shiftr}(S)(M_{\text{hd}}[0]) &= T \quad \text{if } S(M_{\text{hd}}[0]) = T, \\
\text{shiftr}(S)(M_{\text{hd}}[0]) &= S(M_{\text{hd}}[1]) \quad \text{if } S(M_{\text{hd}}[0]) = F, \\
\text{shiftr}(S)(M_{\text{hd}}[n + 1]) &= S(M_{\text{hd}}[n + 2]).
\end{align*}
\]

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

\[
\begin{align*}
CT_n &= (\text{nt}(n + 1) \circ CT'_n) \leq \text{testhp} : n \geq (CT_n \leq \text{testem} \geq S), \\
CT'_n &= (\text{exshb} : n \circ CT''_n) \leq \text{fetch} \geq (S \leq \text{halt} \geq D), \\
CT''_n &= CT'_n \leq \text{testhp} : n \geq (CT''_n \leq \text{testem} \geq S).
\end{align*}
\]

Moreover, take the function $\phi$ from $\mathbb{N}$ to $T_{\text{finrec}}$ defined by $\phi(n) = (CT_n)(CE_n)$ as the thread forking function. Then applying Turing thread $p$ to the Maurer machine $TM$ from some state of $TM$ in which the head position is 0 has the same effect as applying $\|_t((CT_0))$ to the Maurer machine $TM''$ from the corresponding state of $TM''$ in which the thread memory contains the stored graph representation of $p$.

**Theorem 3.** Let $p$ be a Turing thread such that $\text{size}(p) \leq \text{size}(M_{\text{thr}})$, and let $S_0 \in S_{TM}$ and $S_0'' \in S_{TM''}$ be such that $S_0 \upharpoonright (M_{\text{tape}} \cup \{\text{rl}\}) = S_0'' \upharpoonright (M_{\text{tape}} \cup \{\text{rl}\})$, and
Proof. It is easy to see that for all \( S \in S_{TM} \), \( S'' \in S_{TM''} \) and \( n, n' \in \mathbb{N} \) such that \( S \mid (\text{tape} \cup \{\text{rr}\}) = S'' \mid (\text{tape} \cup \{\text{rr}\}) \), \( S(\text{head}) = n \) and \( S''(\text{head}) = n' \):

\[
\begin{align*}
I_{\text{test}}(S)(\text{head}) &= n' \quad \text{iff} \quad I_{\text{test}}(S'')(\text{head}) = n', \\
I_{\text{write}}(S)(\text{head}) &= n' \quad \text{iff} \quad I_{\text{write}}(S'')(\text{head}) = n', \\
I_{\text{move}}(S)(\text{head}) &= n' \quad \text{iff} \quad I_{\text{move}}(S'')(\text{head}) = n'.
\end{align*}
\]

Let \( (\|i\|_0, S_i', S_0') \models_{TM'} (\|\langle \pi_1 \rangle \rightarrow \ldots \rightarrow \langle \pi_i \rangle \rangle, S') \). Then we have for all \( i, j \in [1, l], n \in \mathbb{N} \) and \( s \in B_{\text{tape}} \):

1. (a) if \( p_i \in CT_n \) and \( p_j \in CT_n \), then \( i = j \),
(b) \( \text{nt}(n+1) \circ CT_n' \in p_j \) if \( n = \max\{m \mid \exists k \in [1, l] \cdot p_k \in CT_m\} \);
2. if \( S''(\text{head}[n]) = T \), 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 \( p_{k'} \not\in CT_n' \);
3. if \( S''(\text{head}[n]) = T, p_i \in CT_n, p_i \neq S \) and \( p_i \neq D \), then there exists a \( T'' \in S_{TM''} \) and \( p'_i \in \ldots, p'_{i-1} \in p_i-1 \) such that \( (\|\langle \langle p_1 \rangle \rightarrow \ldots \rightarrow \langle p_i \rangle \rangle, S'', p''_{\text{head}[n]} = T''(\langle \langle p_1 \rangle \rightarrow \ldots \rightarrow \langle p_i \rangle \rangle, (\langle \langle p_1 \rangle \rightarrow \ldots \rightarrow \langle p_i \rangle \rangle), T''(\text{head}[n]) = T''(\text{head}[n]) = T' \);  
4. if \( S''(\text{head}[n]) = T, p_i \equiv CT_n \) and \( S''(\text{head}[n]) = \emptyset, S''(\text{head}[n]) = \emptyset \) \( \langle S, D \rangle \), then:
   (a) if \( p_2 \in (\text{head}[n]) = \text{test}_s \), then there exists a \( T'' \in S_{TM''} \) such that \( (\|\langle \langle p_1 \rangle \rightarrow \ldots \rightarrow \langle p_i \rangle \rangle, S'', p''_{\text{head}[n]} = T''(\langle \langle p_1 \rangle \rightarrow \ldots \rightarrow \langle p_i \rangle \rangle, (\langle \langle p_1 \rangle \rightarrow \ldots \rightarrow \langle p_i \rangle \rangle), T''(\text{head}[n]) = T''(\text{head}[n]) = T' \);  
   (b) if \( p_2 \in (\text{head}[n]) = \text{write}_s \), then there exists a \( T'' \in S_{TM''} \) such that \( (\|\langle \langle p_1 \rangle \rightarrow \ldots \rightarrow \langle p_i \rangle \rangle, S'', p''_{\text{head}[n]} = T''(\langle \langle p_1 \rangle \rightarrow \ldots \rightarrow \langle p_i \rangle \rangle, (\langle \langle p_1 \rangle \rightarrow \ldots \rightarrow \langle p_i \rangle \rangle), T''(\text{head}[n]) = T''(\text{head}[n]) = T' \);  
   (c) if \( p_2 \in (\text{head}[n]) = \text{mover} \), then there exists a \( T'' \in S_{TM''} \) and \( p'_i \in p_i \) such that \( (\|\langle \langle p_1 \rangle \rightarrow \ldots \rightarrow \langle p_i \rangle \rangle, S'', p''_{\text{head}[n]} = T''(\langle \langle p_1 \rangle \rightarrow \ldots \rightarrow \langle p_i \rangle \rangle, (\langle \langle p_1 \rangle \rightarrow \ldots \rightarrow \langle p_i \rangle \rangle), T''(\text{head}[n]) = T''(\text{head}[n]) = T' \);
(d) if \( \pi_2(S''(\text{thd}[\text{null}(S'', S''(rr)))) = \text{movel} \), then there exist a \( T'' \in S_{TM''} \) and \( p'_2 \in p_2 \), \( \ldots \), \( p'_1 \in p_1 \) such that \( (\|_1(\langle p_1 \rangle \cdots \langle p_1 \rangle), S'') \vdash_{TM''}^* (\|_1(\langle p'_2 \rangle \cdots \langle p'_1 \rangle \langle CT'' \rangle), T'') \) and \( T'' \upharpoonright (\text{Mtape} \cup \{rr\}) = \text{movel}(S'') \) \( (\text{Mtape} \cup \{rr\}) \) and either \( n > 0 \) and \( T''(\text{Mhd}[n-1]) = T \) or \( n = 0 \) and \( T''(\text{Mhd}[n]) = T \).

6. if \( S''(\text{Mhd}[n]) = T \), \( p_1 \equiv CT'' \), and \( S''(\text{Mthd}[\text{null}(S'', S''(rr)))) \in \{S, D\} \), then:
(a) if \( S''(\text{Mthd}[\text{null}(S'', S''(rr)))) = S \), then there exist a \( T'' \in S_{TM''} \) such that \( (\|_1(\langle p_1 \rangle \cdots \langle p_1 \rangle), S'') \vdash_{TM''}^* (S, T'') \) and \( T'' \upharpoonright \text{Mtape} = S'' \upharpoonright \text{Mtape} \) and \( T''(\text{Mhd}[n]) = T \);
(b) if \( S''(\text{Mthd}[\text{null}(S'', S''(rr)))) = D \), then there exist a \( T'' \in S_{TM''} \) such that \( (\|_1(\langle p_1 \rangle \cdots \langle p_1 \rangle), S'') \vdash_{TM''}^* (D, T'') \) and \( T'' \upharpoonright \text{Mtape} = S'' \upharpoonright \text{Mtape} \) and \( T''(\text{Mhd}[n]) = T \);

7. if \( S''(\text{Mhd}[n]) = T \) and \( p_1 \equiv CT'' \), then there exist a \( T'' \in S_{TM''} \) and \( p'_2 \in p_2 \), \( \ldots \), \( p'_1 \in p_1 \) such that \( (\|_1(\langle p_1 \rangle \cdots \langle p_1 \rangle), S'') \vdash_{TM''}^* (\|_1(\langle CT'' \rangle \langle p'_2 \rangle \cdots \langle p'_1 \rangle), T'') \) and \( T'' \upharpoonright \text{Mtape} = S'' \upharpoonright \text{Mtape} \) and \( T''(\text{Mhd}[n]) = T \).

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 \( p \cdot_{TM} S_0 \), let \((p'_m, S''_m)\) be the first element in the full path of \( \|_1(\langle CT_0 \rangle) \cdot_{TM''} S''_0 \), and let \((p''_{m+1}, S'''_{m+1})\) be the element in the full path of \( \|_1(\langle CT_0 \rangle) \cdot_{TM''} S''_0 \) that follows the \( m+1\)-th element of which the first component equals \( \|_1(\text{exsb}_{\text{fin}} \circ \langle CT'' \rangle) \circ \alpha \) for some \( n \in \mathbb{N} \) and \( \alpha \in T_{\text{move}}^* \). 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 \( S_{\text{thd}}(p) \) to which \( \text{null}(S''_m, S''(rr)) \) points;
- \( S_m \upharpoonright (\text{Mtape} \cup \{rr\}) = S''_m \upharpoonright (\text{Mtape} \cup \{rr\}) \) and for all \( n \in \mathbb{N} \), \( S_m(\text{head}) = n \) iff \( S''_m(\text{Mhd}[n]) = T \).

(for \( m < |p \cdot_{TM} S_0| \) if \( p \cdot_{TM} S_0 \) is convergent). From this, the theorem follows immediately. \( \square \)

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 3 and the definition of the apply operator that it remains finite if the computation concerned is convergent.

In Section 8, the instructions used to simulate Turing machines includes instructions with an infinite input region and instructions with an infinite output region. In Section 9, only instructions 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 instructions. By using thread forking in the way described, the execution remains controlled by a finite-state thread if the computation concerned is convergent. 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 [3], 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 $\parallel_s(\cdot)$ is an operator on a thread vector such that for all $\alpha \in T_{\text{finrec}}^*$, $p', p'' \in T_{\text{finrec}}$, $a \in A_{\tau \tau} \setminus NT$ and $n \in \text{dom}(\phi)$:  

\[ \parallel_s(\langle \rangle) = S, \]
\[ \parallel_s(\langle S \rangle \sim \alpha) = \parallel_s(\alpha), \]
\[ \parallel_s(\langle D \rangle \sim \alpha) = S_D(\parallel_s(\alpha)), \]
\[ \exists! \alpha', \alpha'' \in T_{\text{finrec}}^* \bullet (\alpha' \in \text{perm}(\langle p' \rangle \sim \alpha) \land \alpha'' \in \text{perm}(\langle p'' \rangle \sim \alpha) \land \parallel_s(\langle p' \leq a \geq p'' \rangle \sim \alpha) = \parallel_s(\alpha') \leq a \geq \parallel_s(\alpha'')), \]
\[ \exists! \alpha' \in T_{\text{finrec}}^* \bullet (\alpha' \in \text{perm}(\langle p' \rangle \sim \alpha \sim \langle \phi(n) \rangle) \land \parallel_s(\langle p' \leq \text{nt}(n) \triangleright p'' \rangle \sim \alpha) = \tau \circ \parallel_s(\alpha')). \]

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 $\parallel_s(\cdot)$, the axioms can always be given in the following way:  

\[ D^* \] 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 $\text{perm}(\alpha)$ for the set of all permutations of sequence $\alpha$.  

\[ 6 \] 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 $\text{perm}(\alpha)$ for the set of all permutations of sequence $\alpha$.  

23
where \(a\) stands for an arbitrary member of \(A_{\tau a} \setminus \mathcal{NT}\), for unary functions \(pv_s^+\), \(pv_s^-\) and \(pv_s^{+nt(n)}\) on \(\mathcal{T}_{\mathrm{finrec}}^+\) such that, for all \(\alpha \in \mathcal{T}_{\mathrm{finrec}}^+, pv_s^+\alpha \in \operatorname{perm}(\alpha)\), \(pv_s^-\alpha \in \operatorname{perm}(\alpha)\) and \(pv_s^{+nt(n)}\alpha \in \operatorname{perm}(\alpha)\).

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

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

\[
\begin{align*}
\forall i \in [1, n] : p_i &\equiv \alpha_{pp_s^+((\alpha) \circ \cdots \circ (\alpha))}(i) \land \alpha_{pp_s^-((\alpha) \circ \cdots \circ (\alpha))}(i) \\
\forall i \in [1, n] : p_i &\equiv (p_i', \cdots, p_n') \Rightarrow \alpha_{pp_s^+((\alpha) \circ \cdots \circ (\alpha))}(i) \land \\
\forall i \in [1, n] : p_i &\equiv (p_i', \cdots, p_n') \Rightarrow \alpha_{pp_s^-((\alpha) \circ \cdots \circ (\alpha))}(i)
\end{align*}
\]

Auxiliary relations \(\sim_a^+, \sim_a^- \subseteq \mathcal{T}_{\mathrm{finrec}}^+ \times \mathcal{T}_{\mathrm{finrec}}^+,\) for \(a \in A_{\tau a},\) and \(\sim_b^- \subseteq \mathcal{T}_{\mathrm{finrec}}^+ \times \mathcal{T}_{\mathrm{finrec}}^+,\) for \(b \in A \setminus \mathcal{NT},\) are used below to define fairness of basic interleaving strategies. They are defined as follows:

\[
\begin{align*}
\alpha &\sim_a^+ \alpha' \iff \\
\exists p, p' \in \mathcal{T}_{\mathrm{finrec}}, \alpha'' \in \mathcal{T}_{\mathrm{finrec}}^*: \\
(\alpha = (p \preceq a \preceq p') \circ \cdots \circ (p_n') \wedge \alpha' = pv_s^+\alpha((\alpha) \circ \cdots \circ (\alpha))) &\quad \text{if } a \not\in \mathcal{NT}, \\
\alpha &\sim_a^{+nt(n)} \alpha' \iff \\
\exists p, p' \in \mathcal{T}_{\mathrm{finrec}}, \alpha'' \in \mathcal{T}_{\mathrm{finrec}}^*: \\
(\alpha = (p \preceq nt(n) \preceq p') \circ \cdots \circ (p_n') \wedge \alpha' = pv_s^{+nt(n)}\alpha((\alpha) \circ \cdots \circ (\alpha))) , \\
\alpha &\sim_b^- \alpha' \iff \\
\exists p, p' \in \mathcal{T}_{\mathrm{finrec}}, \alpha'' \in \mathcal{T}_{\mathrm{finrec}}^*: \\
(\alpha = (p \preceq b \preceq p') \circ \cdots \circ (p_n') \wedge \alpha' = pv_s^-\alpha((\alpha) \circ \cdots \circ (\alpha))) .
\end{align*}
\]

Let \(\|_s(\_\_\_\_\_\_\_\_\_\_)\) be a basic strategic interleaving operator. Then \(\|_s(\_\_\_\_\_\_\_\_\_\_)\) is \(\textit{fair}\) if for all \(a_0, a_1, \ldots \in \mathcal{T}_{\mathrm{finrec}}^+\) and \(j_0 \in [1, |a_0|], j_1 \in [1, |a_1|], \ldots :\)

\[
\begin{align*}
\bigwedge\{\exists a \in A_{\tau a} : (a_i \sim_a^+ a_{i+1} \wedge pp_s^+\alpha_i)(j_i) = j_{i+1} &\quad \forall i \in \mathbb{N}, \exists a \in A \setminus \mathcal{NT} : (a_i \sim_b^- a_{i+1} \wedge pp_s^-\alpha_i)(j_i) = j_{i+1} \} \Rightarrow \bigvee_{i \in \mathbb{N}} j_{i+1} = 1.
\end{align*}
\]
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 4.** Theorem 3 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 3. 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 $S''(M_{td}[n]) = T$ and $p_1 \equiv CT_n$, then there exist a $T'' \in S_{TM''}$ and $p_2' \in p_2$, ..., $p_i' \in p_i$ such that $(||\{p_1\} \circ \ldots \circ \{p_i\}||, S'') \Rightarrow_{TM''} (||\{CT_{n+1} \circ \ldots \circ \{p_i\}||, T'')$ and $T''\mid M_{tape} = S''\mid M_{tape}$ and $T''\mid (M_{td}[n]) = T$.

5’. if $S''(M_{td}[n]) = T$, $p_1 \equiv CT'_n$, and $S''(M_{td}[nlt(S'', S'_n))] \notin \{S, D\}$, then:

(a) if $\pi_2(S''(M_{td}[nlt(S'', S'_n)])) = test$, then there exist a $T'' \in S_{TM''}$ and $p_2' \in p_2$, ..., $p_i' \in p_i$ such that $(||\{p_1\} \circ \ldots \circ \{p_i\}||, S'') \Rightarrow_{TM''} (||\{CT'_n \circ \ldots \circ \{p_i\}||, T'')$ and $T''\mid (M_{tape} \cup \{\tau\}) = I_{test\cdot n}(S'')\mid (M_{tape} \cup \{\tau\})$ and $T''\mid (M_{td}[n]) = T$.

(b) if $\pi_2(S''(M_{td}[nlt(S'', S'_n)])) = write\cdot s$, then there exist a $T'' \in S_{TM''}$ and $p_2' \in p_2$, ..., $p_i' \in p_i$ such that $(||\{p_1\} \circ \ldots \circ \{p_i\}||, S'') \Rightarrow_{TM''} (||\{CT'_n \circ \ldots \circ \{p_i\}||, T'')$ and $T''\mid (M_{tape} \cup \{\tau\}) = I_{write\cdot s\cdot n}(S'')\mid (M_{tape} \cup \{\tau\})$ and $T''\mid (M_{td}[n]) = T$.

These weaker properties are sufficient to complete the proof. 

## 12 Conclusions

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 instructions 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 instructions 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 instructions with a finite input region and a finite output region. This is attained by replacing each instruction with an infinite input region or an infinite output region by a countably infinite number of instructions, 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, an infinite-state thread cannot be regarded as the behaviour of a deterministic sequential program under execution on a real computer.

25
In the third way, the Maurer machine on which Turing machines are simulated has again only instructions 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 instructions. 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 convergent computations.

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

Our interest in Maurer machines originates from our plan to develop, as part of a project investigating micro-threading architectures [10], an approach to design processor architectures that exploit instruction level parallelism. As a first step, we have developed a basic theory of stored threads and their execution based on Maurer machines. The development of that theory is presented in [4]. Like the work presented in this paper, that work also builds upon thread algebra [3]. Our work presented in [5], on a framework for specification and verification of systems that consist of several multi-threaded programs on various hosts in different networks, builds upon thread algebra as well.

References


