Maurer computers with single-thread control

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.
Maurer Computers with Single-Thread Control

J.A. Bergstra$^{1,2}$ and C.A. Middelburg$^3$

$^1$ Programming Research Group, University of Amsterdam, P.O. Box 41882, 1009 DB Amsterdam, the Netherlands
janb@science.uva.nl

$^2$ Department of Philosophy, Utrecht University, P.O. Box 80126, 3508 TC Utrecht, the Netherlands
janb@phil.uu.nl

$^3$ Computing Science Department, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, the Netherlands
keesm@win.tue.nl

Abstract. We present the development of a theory of stored threads and their execution. The work builds upon Maurer’s theory of computer instructions and the thread algebra of Bergstra et al. The theory being developed is primarily relevant to the design of new processor architectures. We also relate Maurer’s model for computers with Turing machines, and stored threads with programs as considered in the program algebra of Bergstra et al.

Keywords: Maurer computers, thread algebra, instructions, stored threads, control threads, Turing machines, program algebra.


1 Introduction

In [11], a paper from almost 40 years ago, Maurer proposes a model for computers that is quite different from the well-known models such as pushdown automata and Turing machines (see e.g. [7]). The strength of Maurer’s model is that it is close to real computers. Computer instructions play a prominent part in his model. We use the phrase Maurer computer for what is a computer according to Maurer’s definition of a computer.

We plan to develop an approach to design processor architectures based on Maurer computers. As a first step, we develop in the current paper a theory of stored threads and their execution. We show that a single thread can control the execution on a Maurer computer of an arbitrary finite-state thread stored in the memory of the Maurer computer, provided that the basic actions of the stored thread correspond to instructions of the Maurer computer. We demonstrate that finite-state threads of arbitrary size can be dealt with if the Maurer computer on which the execution takes place leaves the fetching of the basic actions to another Maurer computer of which the memory size is sufficient for the thread concerned.
To describe threads, we use BTA (Basic Thread Algebra), introduced in [4] 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. Using some kind of strategic interleaving, several single-thread controlled Maurer computers can be put in parallel, which is relevant to the design of new processor architectures. Several kinds of strategic interleaving have been elaborated in earlier work, see e.g. [5]. In this paper, a simple kind of strategic interleaving, called cyclic interleaving, is used to demonstrate that finite-state threads of arbitrary size can be dealt with by setting the fetching of basic actions apart.

A straightforward way to simulate a Turing machine by a Maurer computer clarifies 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. Using Maurer computers, we show that the instructions concerned 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.

We also demonstrate that there is a close connection between stored threads and PGLD programs. PGLD is one of the simpler programming languages based on PGA (ProGram Algebra) introduced in [4]. From each stored thread, a PGLD program can be extracted of which the behaviour is the thread represented by the stored thread. The PGLD program extracted from a stored thread shows hardly any difference with the stored thread. However, PGLD permits a more efficient representation of threads than the one obtained in this indirect way.

The structure of this paper is as follows. First of all, we review Maurer computers (Section 2) and Basic Thread Algebra (Section 3). Following this, we introduce an operator which allows for threads to transform the states of a Maurer computer by means of its instructions. (Section 4). After that, we enhance Maurer computers step by step till we can show that a single thread can control the execution of any stored finite-state thread (Sections 5–8). We also show that such control can be accomplished with a single control instruction (Section 9). Next, we introduce parallel composition of Maurer computers and cyclic interleaving of threads (Section 10) and demonstrate that finite-state threads of arbitrary size can be dealt with by setting the fetching of basic actions apart (Section 11). Then, we clarify some interesting points about Turing machines using Maurer computers (Section 12), and relate PGLD programs to stored threads (Section 13). Finally, we make some concluding remarks (Section 14).

2 Maurer Computers

In this section, we shortly review Maurer computers, i.e. computers as defined by Maurer in [11]. The proofs of the presented theorems can be found in [10].
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.

The following proposition gives an interesting characterization of the set of states of a Maurer computer.

**Proposition 1 (Characterization of the set of states).** Let $(M, B, S, \mathcal{I})$ be a Maurer computer, let $S_0 \in S$, and let $B_x = \{b \in B \mid \exists S \in S \cdot S(x) = b\}$ for all $x \in M$. Then $S$ is the set of all functions $S : M \to B$ such that $S(x) \in B_x$ for $x \in M$ and $\{x \in M \mid S_0(x) \neq S(x)\}$ is finite.

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 $I$. We have the following theorem about the relation between the input region and output region of an instruction.

**Theorem 1 (Input and output regions of instructions).** Let $(M, B, S, \mathcal{I})$ be a Maurer computer, let $S_1, S_2 \in S$, and let $I \in \mathcal{I}$. Then $S_1 \upharpoonright IR(I) = S_2 \upharpoonright IR(I)$ implies $I(S_1) \upharpoonright OR(I) = I(S_2) \upharpoonright OR(I)$.

Both conditions in the definition of Maurer computers are necessary for Theorem 1 to hold.

We have the following theorem about the input region and the output region of the composition of two instructions.
Theorem 2 (Composition of instructions). Let \((M, B, S, I)\) be a Maurer computer, let \(I_1, I_2 \in I\), and let \(J : S \to S\) be defined by \(J(S) = I_2(I_1(S))\). Then \(IR(J) \subseteq IR(I_1) \cup IR(I_2)\) and \(OR(I_1) \setminus OR(I_2) \subseteq OR(J) \subseteq OR(I_1) \cup OR(I_2)\). If \(OR(I_1) \cap IR(I_2) = \emptyset\), then \(IR(I_2) \subseteq IR(J)\) and \(OR(J) = OR(I_1) \cup OR(I_2)\). Moreover, if \(OR(J) = OR(I_1) \cup OR(I_2)\) and \(OR(I_1) \cap OR(I_2) = \emptyset\), then also \(IR(I_1) \subseteq IR(J)\). If \(OR(I_1) \cap IR(I_2) = \emptyset\), \(IR(I_1) \cap OR(I_2) = \emptyset\) and \(OR(I_1) \cap OR(I_2) = \emptyset\), then \(J = J'\) where \(J' : S \to S\) is defined by \(J'(S) = I_1(I_2(S))\).

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

Theorem 3 (Decomposition of instructions). Let \((M, B, S, I)\) be a Maurer computer, let \(I \in I\), and let \(x \in OR(I) \setminus IR(I)\). Then there exist \(J_1, J_2 : S \to S\) with \(J_2(J_1(S)) = I(S)\) such that \(IR(J_1) \subseteq IR(I)\), \(IR(J_2) \subseteq IR(I)\), \(OR(J_1) = \{x\}\) and \(OR(J_2) = OR(I) \setminus \{x\}\).

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

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

Theorem 4 (Existence of instructions (1)). Let \((M, B, S, I)\) be a Maurer computer, let \(Z\) be its unit component, and let \(P, Q \subseteq M\). Then there exists a function \(I : S \to S\) with \(IR(I) = P\) and \(OR(I) = Q\) iff \(P \cap Z = \emptyset, Q \cap Z = \emptyset,\) and \(P \neq \emptyset \Rightarrow Q \neq \emptyset\).

Let \((M, B, S, T)\) be a Maurer computer, let \(I \in I\), 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 \(I\). We have the following theorem about the existence of instructions for arbitrary output, input and affected regions.

Theorem 5 (Existence of instructions (2)). Let \((M, B, S, I)\) be a Maurer computer with countable \(M\), let \(Z\) be its unit component, let \(P, Q \subseteq M\) with \(P \cap Z = \emptyset, Q \cap Z = \emptyset,\) and \(P \neq \emptyset \Rightarrow Q \neq \emptyset,\) and let \(Q_x \subseteq Q\) with \(Q_x \neq \emptyset\) for each \(x \in P\). Moreover, assume that the following two conditions are satisfied:

- there exist only finitely many \(x \in M\) such that \(x \in Q_x, y \notin Q_y\) for all \(y \in M \setminus \{x\}\), and \(\text{card}\{b \in B \mid \exists S \in S \cdot S(x) = b\} = 2\);
- for all infinite \(Q_0 \subseteq \bigcup_{x \in P} Q_x\), the set \(\{x \in P \mid Q_x \cap Q_0 \neq \emptyset\}\) is either infinite or contains an element \(y\) for which the set \(\{b \in B \mid \exists S \in S \cdot S(y) = b\}\) is infinite.
Then there exists a function \( I : S \to S \) with \( IR(I) = P \), \( OR(I) = Q \) and \( AR(\{x\}, I) = Q_x \) for each \( x \in P \).

Both conditions in Theorem 5 are satisfied if \( \bigcup_{x \in P} Q_x \) is a finite set.

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 \( \mathcal{A} \). BTA has the following constants and operators:

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

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).

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 \( V \). 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 over BTA 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 the recursive specification. In the projective limit model of BTA, which is presented in [2, 4], guarded recursive specifications have unique solutions. A thread that is the solution of a finite guarded recursive specification over BTA is called a finite-state thread. BTA with recursion has, in addition to the constants and operators of BTA, for each guarded recursive specification \( E \) over BTA and each variable \( X \) that occurs as the left-hand side of an equation in \( E \), a constant standing for the unique solution of \( E \) for \( X \), which is denoted by \( \langle X | E \rangle \). 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. Following [2, 4], approximation of depth \( n \) is phrased in terms of a unary projection operator \( \pi_n(\_\_ \_ \_) \). The projection operators are defined inductively by means of the axioms in Table 1. In this table, and all subsequent tables with axioms in which \( a \) occurs, \( a \) stands for an arbitrary basic action from \( \mathcal{A} \).

In the structural operational semantics, we represent an execution environment by a function \( \rho: \mathcal{A}^+ \rightarrow \{T, F\} \). We write \( \mathcal{E} \) for the set of all those functions and \( \mathcal{B} \) for the set \( \{T, F\} \). Given an execution environment \( \rho \) and a basic action \( a \), the derived execution environment \( \partial_a \rho \) is defined by \( \partial_a \rho(p) = \rho([a] \bowtie p) \).

The chosen representation of execution environments is based on the assumption that it depends at any stage only on their history, i.e. the sequence of basic actions processed before, whether the reply is positive or negative. This is a realistic assumption for deterministic execution environments.

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

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

These transition relations can be explained as follows:

- \( \langle p, \rho \rangle \xrightarrow{a} \langle p', \rho' \rangle \): in execution environment \( \rho \), thread \( p \) is capable of first performing basic action \( a \) and then proceeding as thread \( p' \) in execution environment \( \rho' \);
- \( \langle p, \rho \rangle \downarrow \): in execution environment \( \rho \), thread \( p \) is capable of terminating successfully;
- \( \langle p, \rho \rangle \uparrow \): in execution environment \( \rho \), thread \( p \) is neither capable of performing a basic action nor capable of terminating successfully.

The structural operational semantics of BTA with recursion and projection is described by the transition rules given in Table 2. In this table, and all subsequent tables with transition rules in which \( a \) occurs, \( a \) stands for an arbitrary basic action from \( \mathcal{A} \). We write \( \langle t | E \rangle \) for \( t \) with, for all \( X \) that occur on the left-hand side of an equation in \( E \), all occurrences of \( X \) in \( t \) replaced by \( \langle X | E \rangle \).

\[ \begin{align*}
\pi_0(x) &= D & \text{P0} \\
\pi_{n+1}(S) &= S & \text{P1} \\
\pi_{n+1}(D) &= D & \text{P2} \\
\pi_{n+1}(x \preceq a \succeq y) &= \pi_n(x) \preceq a \succeq \pi_n(y) & \text{P3} \\
(\bigwedge_{n \geq 0} \pi_n(x) = \pi_n(y)) & \Rightarrow x = y & \text{AIP}
\end{align*} \]

\[ \begin{align*}
\pi_0(x) &= D & \text{P0} \\
\pi_{n+1}(S) &= S & \text{P1} \\
\pi_{n+1}(D) &= D & \text{P2} \\
\pi_{n+1}(x \preceq a \succeq y) &= \pi_n(x) \preceq a \succeq \pi_n(y) & \text{P3} \\
(\bigwedge_{n \geq 0} \pi_n(x) = \pi_n(y)) & \Rightarrow x = y & \text{AIP}
\end{align*} \]
Table 2. Transition rules for BTA with recursion and projection

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (S, \rho) \downarrow )</td>
<td>( (D, \rho) \uparrow )</td>
</tr>
<tr>
<td>( \langle x \leq a \geq y, \rho \rangle \xrightarrow{\Delta} \langle x, \frac{\partial}{\partial a} \rho \rangle )</td>
<td>( \rho(a) = T )</td>
</tr>
<tr>
<td>( \langle y, \frac{\partial}{\partial a} \rho \rangle )</td>
<td>( \rho(a) = F )</td>
</tr>
<tr>
<td>( \langle {t</td>
<td>E}, \rho \rangle \xrightarrow{a} \langle x', \rho' \rangle )</td>
</tr>
<tr>
<td>( \langle X(E), \rho \rangle \xrightarrow{a} \langle x', \rho' \rangle )</td>
<td>( \langle X(E), \rho \rangle \downarrow )</td>
</tr>
<tr>
<td>( \langle x, \rho \rangle \xrightarrow{\pi_{n+1}} \langle \pi_{n}(x), \rho' \rangle )</td>
<td>( \langle \pi_{n}(x), \rho \rangle \downarrow )</td>
</tr>
<tr>
<td>( \langle x, \rho \rangle \xrightarrow{\pi_{n}} \langle \pi_{n+1}(x), \rho' \rangle )</td>
<td>( \langle \pi_{n+1}(x), \rho \rangle \uparrow )</td>
</tr>
</tbody>
</table>

Bisimulation equivalence is defined as follows. A bisimulation is a symmetric binary relation \( B \) on closed terms such that for all closed terms \( p \) and \( q \):
- if \( B(p, q) \) and \( \langle p, \rho \rangle \xrightarrow{a} \langle q', \rho' \rangle \), then there is a \( q' \) such that \( \langle q, \rho \rangle \xrightarrow{a} \langle q', \rho' \rangle \);
- if \( B(p, q) \) and \( \langle p, \rho \rangle \downarrow \), then \( \langle q, \rho \rangle \downarrow \);
- if \( B(p, q) \) and \( \langle p, \rho \rangle \uparrow \), then \( \langle q, \rho \rangle \uparrow \).

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

Bisimulation equivalence is a congruence with respect to all operators involved. This follows from the fact that the transition rules from Table 2 constitute a transition system specification in path format (see e.g. [1]).

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. Moreover, we write \( T_{\text{finrec}}(A) \), where \( A \subseteq A \), for the set of all closed terms from \( T_{\text{finrec}} \) that only contain basic actions from \( A \).

4 Applying Threads to Maurer Machines

In this section, we introduce Maurer machines and add for each Maurer machine \( H \) a binary apply operator \( \cdot H \cdot \) to BTA.

A Maurer machine is a tuple \( H = (M, B, S, I, A, [-]) \), where \( (M, B, S, I) \) is a Maurer computer and:
- \( A \subseteq A \);
- \([-]: A \rightarrow (I \times M)\).

The elements of \( A \) are called the basic actions of \( H \), and \([-]\) 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
Then the resulting state, the reply produced by \( H \) is defined as follows:

\[
S \cdot_H S = S
\]

\[
D \cdot_H S = \dagger
\]

\[
x \cdot_H \dagger = \dagger
\]

\[
(x \preceq a \succeq y) \cdot_H S = x \cdot_H I_a(S) \quad \text{if } I_a(S)(m_a) = T
\]

\[
(x \preceq a \succeq y) \cdot_H S = y \cdot_H I_a(S) \quad \text{if } I_a(S)(m_a) = F
\]

**Table 3.** Defining equations for apply operator

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S \cdot_H S = S )</td>
<td>( S \cdot_H S = S )</td>
</tr>
<tr>
<td>( D \cdot_H S = \dagger )</td>
<td>( D \cdot_H S = \dagger )</td>
</tr>
<tr>
<td>( x \cdot_H \dagger = \dagger )</td>
<td>( x \cdot_H \dagger = \dagger )</td>
</tr>
<tr>
<td>( (x \preceq a \succeq y) \cdot_H S = x \cdot_H I_a(S) )</td>
<td>( (x \preceq a \succeq y) \cdot_H S = x \cdot_H I_a(S) ) if ( I_a(S)(m_a) = T )</td>
</tr>
<tr>
<td>( (x \preceq a \succeq y) \cdot_H S = y \cdot_H I_a(S) )</td>
<td>( (x \preceq a \succeq y) \cdot_H S = y \cdot_H I_a(S) ) if ( I_a(S)(m_a) = F )</td>
</tr>
</tbody>
</table>

**Table 4.** Rule for divergence

\[
\forall n \in \mathbb{N} \cdot \pi_n(x) \cdot_H S = \dagger
\]

\[
x \cdot_H S = \dagger
\]

associated Maurer machine by means of its instructions. Such state transformations produce either a state of the associated Maurer machine or the *undefined state* \( \dagger \). It is assumed that \( \dagger \) is not a state of any Maurer machine. We extend function restriction to \( \dagger \) by stipulating that \( \dagger \dagger M = \dagger \) for any set \( M \).

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

Let \( H = (M, B, S, \mathcal{I}, A, [\cdot]) \) be a Maurer machine, let \( p \in \mathcal{T}_{\text{finrec}}(A) \), and let \( S \in S \). Then \( p \cdot_H S \) 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 \( (I_a, m_a) = [\cdot] \) 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 \( \dagger \).

Let \( H = (M, B, S, \mathcal{I}, A, [\cdot]) \) be a Maurer machine, and let \( (I_a, m_a) = [\cdot] \) for all \( a \in A \). Then the apply operator \( \cdot_H \cdot \) is defined by the equations given in Table 3 (for \( a \in A \) and \( S \in S \)) and the rule given in Table 4 (for \( S \in S \)). We say that \( p \cdot_H S \) is *convergent* if \( \exists n \in \mathbb{N} \cdot \pi_n(p) \cdot_H S \neq \dagger \). If \( p \cdot_H S \) is convergent, then the *number of computation steps* of \( p \cdot_H S \), written \( |p \cdot_H S| \), is the smallest \( n \in \mathbb{N} \) such that \( \pi_n(p) \cdot_H S \neq \dagger \). If \( p \cdot_H S \) is not convergent, then \( |p \cdot_H S| \) is undefined. We say that \( p \cdot_H S \) is *divergent* if \( p \cdot_H S \) is not convergent. Notice that the rule from Table 4 can be read as follows: if \( x \cdot_H S \) is divergent, then it equals \( \dagger \).

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

- if \( I_a(S)(m_a) = T \) and \( p = p' \preceq a \succeq p'' \), then \( (p, S) \vdash_H (p', I_a(S)) \);
- if \( I_a(S)(m_a) = F \) and \( p = p' \preceq a \succeq p'' \), then \( (p, S) \vdash_H (p'', I_a(S)) \).
If \((p, S) \vdash_H (p', S')\), then \(p \cdot_H S = p' \cdot_H S'\). Moreover, let \(p \in T_{\text{store}}(A)\), and let \(S \in \mathcal{S}\). Then the full path of \(p \cdot_H S\) is the unique full path in \(\vdash_H \cdot\) from \((p, S)\).

A full path in \(\vdash_H \cdot\) is one of the following:

- a finite path \(((p_0, S_0), \ldots, (p_n, S_n))\) in \(\vdash_H \cdot\) such that there exists no \((p_{n+1}, S_{n+1}) \in T_{\text{store}}(A) \times \mathcal{S}\) with \((p_n, S_n) \vdash_H (p_{n+1}, S_{n+1})\);
- an infinite path \(((p_0, S_0), (p_1, S_1), \ldots)\) in \(\vdash_H \cdot\).

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

5 Executing Stored Basic Actions

In this section, we enhance Maurer machines such that processing of a basic action performed by a thread amounts to first storing it in a special memory element and then executing the instruction associated with the basic action stored in that special memory element. That is, we extend the memory with a basic action register (\(\text{bar}\)) and a reply register (\(\text{rr}\)), and the instruction set with store instructions for each action \(a\) of the original Maurer machine \((I_{\text{store,}a})\) and an execute stored basic action instruction \((I_{\text{exsba}})\). Moreover, we replace the basic actions of the original Maurer machine by basic actions with which the extra instructions are associated. The resulting Maurer machines are called SBA-enhancements. SBA stands for stored basic action.

Let \(A \subseteq A\) be such that for all \(a \in A\), \(\text{store:a} \notin A\). Then it is assumed that \(\text{store:a} \in A\) for all \(a \in A\). Moreover, it is assumed that \(\text{exsba} \in A\).

Let \(H = (M, B, S, T, A, [-])\) be a Maurer machine with \(\text{bar}\), \(\text{rr} \notin M\), \(I_{\text{store,}a} \notin T\) for all \(a \in A\), \(I_{\text{exsba}} \notin T\), \(\text{store:a} \notin A\) for all \(a \in A\) and \(\text{exsba} \notin A\), and let \((I_a, m_a) = [a]\) for all \(a \in A\). Then the SBA-enhancement of \(H\) is the Maurer machine \((M', B', S', T', A', [-'])\) such that

\[
M' = M \cup \{\text{bar}, \text{rr}\},
B' = B \cup A \cup \mathbb{B},
S' = \{S' : M' \rightarrow B' \mid S' \in S \land S'(\text{bar}) \in A \land S'(\text{rr}) \notin \mathbb{B}\},
T' = \{I' : S' \rightarrow S' \mid \exists I \in T \forall S' \in S' \cdot
ing (I'(S')) \upharpoonright M = I(S' \upharpoonright M) \land I'(S') \upharpoonright (M' \setminus M) = S' \upharpoonright (M' \setminus M)\}
\cup \{I_{\text{store,}a} \mid a \in A\} \cup \{I_{\text{exsba}}\},
A' = \{\text{store:a} \mid a \in A\} \cup \{\text{exsba}\},
[a]' = (I_a, \text{rr}) \text{ for all } a \in A'.
\]

Here, for each \(a \in A\), \(I_{\text{store,}a}\) is the unique function from \(S'\) to \(S'\) such that for all \(S' \in S'\):

\[
I_{\text{store,}a}(S') \upharpoonright M = S' \upharpoonright M,
I_{\text{store,}a}(S')(\text{bar}) = a,
I_{\text{store,}a}(S')(\text{rr}) = S'(\text{rr}).
\]
and \( I_{\text{esba}} \) is the unique function from \( S' \) to \( S' \) such that for all \( S' \in S' \):

\[
\begin{align*}
I_{\text{esba}}(S') | M & = I_{\text{SBA}(\text{bar})}(S' | M), \\
I_{\text{esba}}(S')(\text{bar}) & = S'(\text{bar}), \\
I_{\text{esba}}(S')(\text{rr}) & = I_{\text{SBA}(\text{bar})}(S' | M)(m_{S'/(\text{bar})}).
\end{align*}
\]

Because the memory is extended with only finitely many memory elements, it is easy to check, using Proposition 1, that an SBA-enhancement of a Maurer machine is a Maurer machine indeed. The same remark applies to all subsequent enhancements as well.

We define inductively a transformation function \( \phi \) on \( T_{\text{finrec}} \):

\[
\begin{align*}
\phi(X) & = X, \\
\phi(S) & = S, \\
\phi(D) & = D, \\
\phi(t_1 \sqsubseteq a \sqsupseteq t_2) & = \text{store}:a \circ (\phi(t_1) \sqsubseteq \text{esba} \sqsupseteq \phi(t_2)), \\
\phi((X_0[|X_0 = t_0, \ldots, X_n = t_n])) & = (X_0[|X_0 = \phi(t_0), \ldots, X_n = \phi(t_n))\).
\end{align*}
\]

Applying thread \( p \) to a state of Maurer machine \( H \) has the same effect as applying the transformation of \( p \) to the corresponding state of the SBA-enhancement of \( H \).

Theorem 6 (SBA-enhancement). Let \( H' = (M', B', S', T', A', [\cdot]) \) be the SBA-enhancement of \( H = (M, B, S, T, A, [\cdot]) \), let \( S' \in S' \), and let \( p \in T_{\text{finrec}}(A) \). Then \( p \bullet_H (S'_0 | M) = (\phi(p) \bullet_{H'} S'_{0}) | M \).

Proof. Let \( (I_a, m_a) = [a] \) for all \( a \in A \), and let \( (I_{a'}, m_a') = [a'] \) for all \( a' \in A' \). It is easy to see that for all \( a \in A \) and \( S' \in S' \):

\[
\begin{align*}
I_a(S' | M) & = I_{\text{esba}}(I_{\text{store}:a}(S')) | M, \\
I_a(S' | M)(m_a) & = I_{\text{esba}}(I_{\text{store}:a}(S'))(\text{rr}).
\end{align*}
\]

In the case where \( p \bullet_H (S'_0 | M) \) is convergent, it is easy to prove the theorem by induction on \([p \bullet_H (S'_0 | M)]\), using equations (1) and (2). In the case where \( p \bullet_H (S'_0 | M) \) is not convergent, the theorem follows immediately from the claim that \( \pi_n(p) \bullet_H (S'_0 | M) = (\pi_{2n}(\phi(p)) \bullet_{H'} S'_{0}) | M \) for all \( n \in \mathbb{N} \). This claim is easily proved by induction on \( n \), using equations (1) and (2). \( \Box \)

On the SBA-enhancement of a Maurer machine \( H \), processing of a basic action performed by a thread \( p \) amounts to first storing it in the special memory element \( \text{bar} \) and then executing the instruction associated with the basic action stored in \( \text{bar} \). For storing basic actions in \( \text{bar} \) and executing basic actions stored in \( \text{bar} \), the special basic actions \( \text{store}:a \) and \( \text{esba} \) are introduced. Thus, processing can be brought under control of a variant of the thread \( p \), viz. the thread obtained by applying the transformation \( \phi \) to \( p \).

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

6 Representation of Threads

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

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

The thread memory is a memory of which the elements can be addressed by means of elements of \([0, \text{size}(M_t) - 1]\). We write \( A_t \) for \([0, \text{size}(M_t) - 1]\).

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

Let \( n, n' \in A_t \) be such that \( n \leq n' \). Then, we write \( B_t[n, n'] \) for \( (S, D) \cup ([n, n'] \times A) \). We write \( B_t \) for \( B_t[0, \text{size}(M_t) - 1]\). \( B_t \) is called the thread memory base set. We write \( S_t \) for the set of all functions \( S_t : M_t \to B_t \).

Let \( p \in T_{\text{finrec}} \). Then the nodes of the graph representation of \( p \), written \( \text{Nodes}(p) \), is the smallest subset of \( T_{\text{finrec}} \) such that:

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

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

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

Let \( p \in T_{\text{finrec}} \) be such that \( \text{size}(p) \leq \text{size}(M_t) \). Then the stored graph representation of \( p \), written \( s_t(p) \), is the unique \( s_t : M_t[0, \text{size}(p) - 1] \to B_t[0, \text{size}(p) - 1] \) such that for all \( n \in [0, \text{size}(p) - 1] \), \( s_t(M_t[n]) = \text{nprepr}_p(\text{node}_p(n)) \), where \( \text{nprepr}_p : \text{Nodes}(p) \to B_t[0, \text{size}(p) - 1] \) is defined as follows:

\[
\begin{align*}
\text{nprepr}_p(S) &= S, \\
\text{nprepr}_p(D) &= D, \\
\text{nprepr}_p(p' \leq a \triangleright q') &= (\text{node}_p^{-1}(p'), a, \text{node}_p^{-1}(q')) , \\
\text{nprepr}_p(\langle X_0 \mid X_0 = t_0, \ldots, X_n = t_n \rangle) &= \text{nprepr}_p(\langle t_0 \mid X_0 = t_0, \ldots, X_n = t_n \rangle).
\end{align*}
\]
We call $s_t(p)$ a stored thread.

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

7 No Stored Threads, but a Single Control Thread

Before we make use of stored threads to accomplish that a single control thread is sufficient, we demonstrate in this section that this can also be accomplished without stored threads at the cost of flexibility.

We enhance Maurer machines by extending the memory with a node register (nr), a basic action register (bar) and a reply register (rr), and the instruction set with a halt instruction ($I_{\text{halt}}$), two fetch instructions ($I_{\text{fetch}}$, $I_{\text{fetch}}$) and an execute stored basic action instruction ($I_{\text{exsba}}$). Moreover, we replace the basic actions of the original Maurer machine by basic actions with which the extra instructions are associated. The resulting Maurer machines are called SBA’-enhancements.

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

It is assumed that halt $\in A$, that fetch:$r \in A$ for all $r \in B$, and that exsba $\in A$.

Let $H = (M, B, S, I, A, \{[\_])$ be a Maurer machine with $nr, bar, rr \not\in M$, $I_{\text{halt}} \notin I$, $I_{\text{fetch}}, r \notin I$ for all $r \in B$, $I_{\text{exsba}} \notin I$, halt $\notin A$, fetch:$r \notin A$ for all $r \in B$ and exsba $\not\in A$, and let $(I_a, m_a) = [a]$ for all $a \in A$. Let also $p \in T_{\text{finite}}(A)$. Then the SBA’-enhancement of $H$ for $p$ is the Maurer machine $H' = (M', B', S', I', A', [\_'])$ such that

$$M' = M \cup \{nr, bar, rr\} ,$$

$$B' = B \cup [-1, \text{size}(p) - 1] \cup A \cup B ,$$

$$S' = \{S' : M' \rightarrow B' | S' \mid M \in S \wedge$$

$$S'(nr) \in [-1, \text{size}(p) - 1] \wedge S'(bar) \in A \wedge S'(rr) \in B\} ,$$

$$I' = \{I' : S' \rightarrow S' | \forall I \in I \bullet S' \in S' \bullet$$

$$I'(S') \mid M = I(S' \mid M) \wedge I'(S') \mid (M' \setminus M) = S' \mid (M' \setminus M)\}$$

$$\cup \{I_{\text{halt}}\} \cup \{I_{\text{fetch}}, r \mid r \in B\} \cup \{I_{\text{exsba}}\} ,$$

$$A' = \{\text{halt}\} \cup \{\text{fetch}:r \mid r \in B\} \cup \{\text{exsba}\} ,$$

$$[a'] = (I_a, rr) \text{ for all } a \in A' .$$

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

$$I_{\text{halt}}(S') \mid M = S' \mid M ,$$

$$I_{\text{halt}}(S')(nr) = S'(nr) ,$$

$$I_{\text{halt}}(S')(bar) = S'(bar) ,$$

$$I_{\text{halt}}(S')(rr) = T \text{ if } \text{node}_p(S'(nr)) = S ,$$

$$I_{\text{halt}}(S')(rr) = F \text{ if } \text{node}_p(S'(nr)) \neq S ;$$
for each $r \in \mathbb{B}$, $I_{\text{fetch},r}$ is the unique function from $S'$ to $S'$ such that for all $S' \in S'$:

$$
I_{\text{fetch},r}(S') \upharpoonright M = S' \upharpoonright M,
I_{\text{fetch},r}(S')(\text{nr}) = \text{nnn}(S', r),
I_{\text{fetch},r}(S')(\text{bar}) = \pi_2(\text{repr}_p(\text{nnn}(S', r))) \text{ if } \text{node}_p(\text{nnn}(S', r)) \notin \{S, D\},
I_{\text{fetch},r}(S')(\text{bar}) = S'(\text{bar}) \text{ if } \text{node}_p(\text{nnn}(S', r)) \in \{S, D\},
I_{\text{fetch},r}(S')(\text{rr}) = T \text{ if } \text{node}_p(\text{nnn}(S', r)) \notin \{S, D\},
I_{\text{fetch},r}(S')(\text{rr}) = F \text{ if } \text{node}_p(\text{nnn}(S', r)) \in \{S, D\},
$$

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

$$
\text{nnn}(S', T) = \pi_1(\text{repr}_p(S'(\text{nr}))) \text{ if } S'(\text{nr}) \neq -1 \land \text{node}_p(S'(\text{nr})) \notin \{S, D\},
\text{nnn}(S', F) = \pi_3(\text{repr}_p(S'(\text{nr}))) \text{ if } S'(\text{nr}) \neq -1 \land \text{node}_p(S'(\text{nr})) \notin \{S, D\},
\text{nnn}(S', r) = S'(\text{nr}) \text{ if } S'(\text{nr}) \neq -1 \land \text{node}_p(S'(\text{nr})) \in \{S, D\},
\text{nnn}(S', r) = 0 \text{ if } S'(\text{nr}) = -1;
$$

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

$$
I_{\text{exsba}}(S') \upharpoonright M = I_{S'(\text{bar})}(S' \upharpoonright M),
I_{\text{exsba}}(S')(\text{nr}) = S'(\text{nr}),
I_{\text{exsba}}(S')(\text{bar}) = S'(\text{bar}),
I_{\text{exsba}}(S')(\text{rr}) = I_{S'(\text{bar})}(S' \upharpoonright M)(m_{S'(\text{bar})}).
$$

The node register $\text{nr}$ is meant for containing the number that corresponds to the node of the graph representation of $p$ from which most recently a basic action has been fetched. That node, together with the reply produced at completion of the execution of the basic action concerned, determines the node from which next time a basic action must be fetched. To indicate that no basic action has been fetched yet, $\text{nr}$ must initially contain $-1$. The number corresponding to the node from which the first time a basic action must be fetched, i.e. the root, is 0.

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

$$
CT = (CT \triangleleft \text{exsba} \triangleright CT') \triangleleft \text{fetch}: T \triangleright (S \triangleleft \text{halt} \triangleright D),
CT' = (CT \triangleleft \text{exsba} \triangleright CT') \triangleleft \text{fetch}: F \triangleright (S \triangleleft \text{halt} \triangleright D).
$$

Applying thread $p$ to a state of Maurer machine $H$ has the same effect as applying control thread $CT$ to the corresponding state of the SBA'-enhancement of $H$ for $p$.

**Theorem 7 (SBA'-enhancement).** Let $H' = (M', B', S', I', A', [], ^{-})$ be the SBA'-enhancement of $H = (M, B, S, I, A, [], ^{-})$ for $p \in T_{\text{node}}(A)$, and let $S'_0 \in S'$ be such that $S'_0(\text{nr}) = -1$. Then $p \bullet_H (S'_0 \upharpoonright M) = (CT' \bullet_H S'_0) \upharpoonright M$. 

13
Proof. Let \((I_a, m_a) = \llbracket a \rrbracket\) for all \(a \in A\), and let \((I_a, \text{rr}) = \llbracket a' \rrbracket\) for all \(a \in A'\). Then it is easy to see that for all \(S' \in S'\) with \(\text{node}_p(mn(S', S'(\text{rr}))) \not\in \{S, \text{D}\}\):

\[
I_a(S' \upharpoonright M) = I_{\text{exsba}}(I_{\text{fetch}, r}(S')) \upharpoonright M, \tag{3}
\]

\[
I_a(S' \upharpoonright M) (m_a) = I_{\text{exsba}}(I_{\text{fetch}, r}(S'))(\text{rr}), \tag{4}
\]

where \(a = \pi_2(\text{repr}_p(\text{node}_p(mn(S', S'(\text{rr}))))\) and \(r = S'(\text{rr})\).

Let \((p_n, S_n)\) be the \(n+1\)-th element in the full path of \(p \cdot H (S_n' \upharpoonright M)\), and let \((p'_n, S'_n)\) be the \(n+1\)-th element in the full path of \(CT \cdot H', S'_0\). Then it is easy to prove by induction on \(n\) that

\[
\begin{align*}
\rho_{2n+2} &= CT & \text{if } S'_{2n+3}(\text{rr}) &= T \\
\rho_{2n+2} &= CT' & \text{if } S'_{2n+3}(\text{rr}) &= \neg T \\
\rho_{2n+2} &= S & \text{if } S'_{2n+3}(\text{rr}) &= F \\
\rho_{2n+2} &= D & \text{if } S'_{2n+3}(\text{rr}) &= \neg F
\end{align*} \tag{5}
\]

(for \(2n + 2 < |CT \cdot H', S_0'|\) if \(CT \cdot H', S_0'\) is convergent). Moreover, using (3), (4) and (5), it is straightforward to prove by induction on \(n\) that:

- \(p_{n+1}\) is represented by the part of the graph representation of \(p\) of which the root is \(\text{node}_p(mn(S_{2n+2}, S'_{2n+3}(\text{rr})))\);
- \(S_{n+1} = S_{2n+2} \upharpoonright M\).

(for \(n + 1 < |p \cdot H (S_0' \upharpoonright M)|\) if \(p \cdot H (S_0' \upharpoonright M)\) is convergent). From this, the theorem follows immediately. \(\square\)

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

8 Fetching Basic Actions from a Stored Thread

In this section, we make use of stored threads to accomplish that a single control thread is sufficient.

We enhance Maurer machines by extending the memory with a thread memory \((M_t)\), a thread location register \((\text{trl})\), a basic action register \((\text{bar})\) and a reply register \((\text{rr})\), and the instruction set with a halt instruction \((I_{\text{halt}})\), two fetch instructions \((I_{\text{fetch}, T}, I_{\text{fetch}, F})\) and an execute stored basic action instruction \((I_{\text{exsba}})\). Moreover, we replace the basic actions of the original Maurer machine by basic actions with which the extra instructions are associated. The resulting Maurer machines are called ST-4I-enhancements. ST stands for stored thread and 4I indicates that there are four control instructions available.

Let \(H = (M, B, S, T, A, \llbracket \_ \rrbracket)\) be a Maurer machine with \(M_t \not\subset M\), \(\text{trl}, \text{bar}, \text{rr} \not\subset M\), \(I_{\text{halt}} \not\subset T\), \(I_{\text{fetch}, r} \not\subset T\) for all \(r \in B\), \(I_{\text{exsba}} \not\subset T\), \(\text{halt} \not\subset A\), \(\text{fetch}:r \not\subset A\) for all \(r \in B\) and \(\text{exsba} \not\subset A\), and let \((I_a, m_a) = \llbracket a \rrbracket\) for all \(a \in A\). Then the ST-4I-
enhancement of $H$ is the Maurer machine $H' = (M', B', S', \mathcal{I}', A', [\_]')$ such that

\[
M' = M \cup M_t \cup \{\text{tlr, bar, } rr\}, \\
B' = B \cup B_t \cup A_t \cup \{-1\} \cup A \cup B, \\
S' = \{S' : M' \to B' | S' \mid M \in S \land S' \mid M_t \in S_t \land \}
\]

\[
S'('tlr) \in A_t \cup \{-1\} \land S'('bar) \in A \land S'('rr) \in B, \\
\mathcal{I}' = \{I' : S' \to S' | \exists I \in \mathcal{I} \land \forall S' \in S' \land \}
\]

\[
I'(S') \mid M = I(S' \mid M) \land I'(S') \mid (M' \setminus M) = S' \mid (M' \setminus M) \\
\}
\]

\[
\cup \{\text{halt} \cup \{\text{fetch}_r | r \in B\} \cup \{\text{exsba}\}, \\
A' = \{\text{halt} \cup \{\text{fetch}_r | r \in B\} \cup \{\text{exsba}\}, \\
[a]' = (\text{halt}_a, r) \text{ for all } a \in A'.
\]

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

\[
I_{\text{halt}}(S') \mid M = S' \mid M, \\
I_{\text{halt}}(S') \mid M_t = S' \mid M_t, \\
I_{\text{halt}}(S')('tlr) = S'('tlr), \\
I_{\text{halt}}(S')('bar) = S'('bar), \\
I_{\text{halt}}(S')('rr) = T \text{ if } S'('tlr) = S, \\
I_{\text{halt}}(S')('rr) = F \text{ if } S'('tlr) \neq S;
\]

for each $r \in B$, $I_{\text{fetch}_r}$ is the unique function from $S'$ to $S'$ such that for all $S' \in S'$:

\[
I_{\text{fetch}_r}(S') \mid M = S' \mid M, \\
I_{\text{fetch}_r}(S') \mid M_t = S' \mid M_t, \\
I_{\text{fetch}_r}(S')('tlr) = \text{ntla}(S', r), \\
I_{\text{fetch}_r}(S')('bar) = \pi_2(S'(M_t[\text{ntla}(S', r)])) \text{ if } S'(M_t[\text{ntla}(S', r)]) \notin \{S, D\}, \\
I_{\text{fetch}_r}(S')('bar) = S'('bar) \text{ if } S'(M_t[\text{ntla}(S', r)]) \in \{S, D\}, \\
I_{\text{fetch}_r}(S')('rr) = T \text{ if } S'(M_t[\text{ntla}(S', r)]) \notin \{S, D\}, \\
I_{\text{fetch}_r}(S')('rr) = F \text{ if } S'(M_t[\text{ntla}(S', r)]) \in \{S, D\},
\]

where $\text{ntla} : S' \times B \to A_t$ is defined as follows:

\[
\text{ntla}(S', T) = \pi_1(S'(M_t[S'('tlr')]))) \text{ if } S'('tlr) \in A_t \land S'(M_t[S'('tlr')]) \notin \{S, D\}, \\
\text{ntla}(S', F) = \pi_3(S'(M_t[S'('bar')])) \text{ if } S'(M_t[\text{ntla}(S', r)]) \notin \{S, D\}, \\
\text{ntla}(S', r) = S'('tlr) \text{ if } S'(M_t[S'('tlr')]) \in \{S, D\}, \\
\text{ntla}(S', r) = 0 \text{ if } S'(M_t[S'('tlr')]) \notin \{S, D\};
\]

15
and $I_{\text{exsba}}$ is the unique function from $S'$ to $S'$ such that for all $S' \in S'$:

\[
I_{\text{exsba}}(S') \upharpoonright M = I_{S'(\text{bar})}(S' \upharpoonright M),
\]
\[
I_{\text{exsba}}(S') \upharpoonright M_k = S' \upharpoonright M_k,
\]
\[
I_{\text{exsba}}(S')(\text{thr}) = S'(\text{thr}),
\]
\[
I_{\text{exsba}}(S')(\text{bar}) = S'(\text{bar}),
\]
\[
I_{\text{exsba}}(S')(\text{rr}) = I_{S'(\text{bar})}(S' \upharpoonright M)(m_{S'(\text{bar})}).
\]

The thread location register $\text{thr}$ 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, $\text{thr}$ 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.

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

\[
CT = (CT \trianglelefteq \text{exsba} \triangleright CT') \trianglelefteq \text{fetch: T} \triangleright (S \trianglelefteq \text{halt} \triangleright D),
\]
\[
CT' = (CT \trianglelefteq \text{exsba} \triangleright CT') \trianglelefteq \text{fetch: F} \triangleright (S \trianglelefteq \text{halt} \triangleright D).
\]

Applying control thread $p$ to a state of Maurer machine $H$ has the same effect as applying control thread $CT$ to the corresponding state of the ST-4I-enhancement of $H$ in which the thread memory contains the stored graph representation of $p$.

**Theorem 8 (ST-4I-enhancement).** Let $H' = (M',B',S',I',A',[\cdot])$ be the ST-4I-enhancement of $H = (M,B,S,I,A,[\cdot])$, let $p \in T_{\text{finite}}(A)$ be such that $\text{size}(p) \leq \text{size}(M_k)$, and let $S'_0 \in S'$ be such that $S'_0 \upharpoonright |M_k[0,\text{size}(p) - 1] = s(p)$ and $S'_0(\text{thr}) = -1$. Then $p \cdot H (S'_0 \upharpoonright M) = (CT \cdot H, S'_0) \upharpoonright M$.

**Proof.** Let $(I_a,m_a) = \llbracket a \rrbracket$ for all $a \in A$, and let $(I_{\text{ntla}},rr) = \llbracket a' \rrbracket$ for all $a' \in A'$. Then it is easy to see that for all $S' \in S'$ with $S'(M_k[\text{ntla}(S',S'(rr))]) \not\in \{S,D\}$:

\[
I_a(S' \upharpoonright M) = I_{\text{exsba}}(I_{\text{fetch:rr}}(S')) \upharpoonright M,
\]
\[
I_a(S' \upharpoonright M)(m_a) = I_{\text{exsba}}(I_{\text{fetch:rr}}(S'))(rr),
\]

where $a = \tau_2(S'(M_k[\text{ntla}(S',S'(rr))]))$ and $r = S'(rr)$.

Let $(p_n,S_n)$ be the $n+1$-th element in the full path of $p \cdot H (S'_0 \upharpoonright M)$, and let $(p'_n,S'_n)$ be the $n+1$-th element in the full path of $CT \cdot H, S'_0$. Then it is easy to prove by induction on $n$ that

\[
p'_{2n+2} = CT \quad \text{if} \quad S'_{2n+1}(rr) = T \quad \text{and} \quad S'_{2n+2}(rr) = T,
\]
\[
p'_{2n+2} = CT' \quad \text{if} \quad S'_{2n+1}(rr) = T \quad \text{and} \quad S'_{2n+2}(rr) = F.
\]
\[
p'_{2n+2} = S \quad \text{if} \quad S'_{2n+1}(rr) = F \quad \text{and} \quad S'_{2n+2}(rr) = T.
\]
\[
p'_{2n+2} = D \quad \text{if} \quad S'_{2n+1}(rr) = F \quad \text{and} \quad S'_{2n+2}(rr) = F.
\]

(for $2n + 2 < |CT \cdot H, S'_0|$ if $CT \cdot H, S'_0$ is convergent). Moreover, using (6), (7) and (8), it is straightforward to prove by induction on $n$ that:
– $p_{n+1}$ is represented by the part of $s_t(p)$ to which $ntla(S_{2n+2}',S_{2n+2}')(rr)$ points;
– $S_{n+1} = S_{2n+2} \upharpoonright M$.

(for $n+1 < |p \bullet_H (S_0' \upharpoonright M)$ if $p \bullet_H (S_0' \upharpoonright M)$ is convergent). From this, the theorem follows immediately.

Notice that the proof of Theorem 7 and the proof of Theorem 8 follow similar lines.

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

9 A Universal Control Instruction

On the ST-4I-enhancements of Maurer machines, four instructions are available for controlling the processing of basic actions. In this section, we enhance Maurer machines such that a single universal control instruction becomes available. That is, we extend the memory with a thread memory ($M_t$), a thread location register ($tlr$), a basic action register ($bar$), a reply register ($rr$) and a fetch mode register ($fmr$), and the instruction set with a step instruction ($I_{\text{step}}$). Moreover, we replace the basic actions of the original Maurer machine by a basic action with which the extra instruction is associated. The resulting Maurer machines are called ST-1I-enhancements. ST stands again for stored thread and 1I indicates that there is one control instruction available.

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

Let $H = (M,B,S,I,A,[\ ])$ be a Maurer machine with $M_t \not\subseteq M$, $tlr, bar, rr, fmr \not\subseteq M$, $I_{\text{step}} \not\in I$ and $\text{step} \not\in A$, and let $(I_a,m_a) = [a]$ for all $a \in A$. Then the ST-1I-enhancement of $H$ is the Maurer machine $H' = (M',B',S',I',A',[\ ]')$ such that

\[
M' = M \cup M_t \cup \{tlr, bar, rr, fmr\},
\]

\[
B' = B \cup B_t \cup A_t \cup \{-1\} \cup A \cup B, \]

\[
S' = \{S' : M' \to B' \mid S' \mid M' \in S \wedge S' \mid M_t \in S_t \wedge
S'(tlr) \in A_t \cup \{-1\} \wedge S'(bar) \in A \wedge S'(rr) \in B \wedge S'(fmr) \in B\}, \]

\[
I' = \{I' : S' \to S' \mid \exists I \in I \bullet S' \in S' \bullet
I'(S') \mid M = I(S' \mid M) \wedge I'(S') \mid (M' \setminus M) = S' \setminus (M' \setminus M)\}
\]

\[
\cup \{I_{\text{step}}\}, \]

\[
A' = \{\text{step}\},
\]

\[
[\text{step}]' = (I_{\text{step}}, rr). \]
Here, $I_{\text{step}}$ is the unique function from $S'$ to $S'$ such that for all $S' \in S'$:

\[
\begin{align*}
I_{\text{step}}(S') \mid M'' & = I_{\text{fetch},r}(S' \mid M'') \quad \text{if } S'(\text{fmr}) = T \land S'(\text{rr}) = r, \\
I_{\text{step}}(S') \mid M'' & = I_{\text{exsba}}(S' \mid M'') \quad \text{if } S'(\text{fmr}) = F \land S'(\text{rr}) = T, \\
I_{\text{step}}(S') \mid M'' & = I_{\text{halt}}(S' \mid M'') \quad \text{if } S'(\text{fmr}) = F \land S'(\text{rr}) = F, \\
I_{\text{step}}(S')(\text{fmr}) & = F \quad \text{if } S'(\text{fmr}) = T, \\
I_{\text{step}}(S')(\text{fmr}) & = T \quad \text{if } S'(\text{fmr}) = F,
\end{align*}
\]

where $M'' = M \cup M_t \cup \{\text{tlr, bar, rr}\}$ and $I_{\text{fetch},r}$, $I_{\text{exsba}}$ and $I_{\text{halt}}$ are defined as in the definition of the ST-4I-enhancement.

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

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

\[
CT'' = (\text{step} \circ CT'') \triangleq \text{step} \triangleright (S \triangleq \text{step} \triangleright D).
\]

Notice that odd steps are actually fetch steps, which may fail because of termination or deadlock of the controlled thread. Applying thread $p$ to a state of Maurer machine $H$ has the same effect as applying control thread $CT''$ to the corresponding state of the ST-1I-enhancement of $H$ in which the thread memory contains the stored graph representation of $p$.

**Theorem 9 (ST-1I-enhancement).** Let $H' = (M', B', S', T', A', [\_])$ be the ST-1I-enhancement of $H = (M, B, S, T, A, [\_])$, let $p \in T_{\text{foreg}}(A)$ be such that $\text{size}(p) \leq \text{size}(M_t)$, and let $S_0' \in S'$ be such that $S_0' \mid M_0[0, \text{size}(p) - 1] = s_t(p)$, $S_0'(\text{tlr}) = -1$ and $S_0'(\text{fmr}) = T$. Then $p \bullet_H (S_0' \mid M) = (CT'' \bullet_H S_0') \mid M$.

**Proof.** The proof follows the same line as the proof of Theorem 8. In the proof, the equations corresponding to equations (6) and (7) hold only for states $S'$ with $S'(\text{fmr}) = T$. This does not stand in the way of following the same line, because this extra condition is satisfied by all states $S'$ that have to be related to the state component of an element in the full path of $p \bullet_H (S_0' \mid M)$.

\]

10 Parallel Maurer Machines and Interleaving of Threads

In Section 11, we will demonstrate that a Maurer machine with a fixed finite memory can deal with any finite-state thread, provided that it is put in parallel with a Maurer machine of a suitable kind that can hold the thread concerned. In this section, we introduce the parallel composition of Maurer machines. Moreover, because the control threads of the Maurer machines have to be interleaved if they are put in parallel, we add an operator for that purpose to BTA.
Table 5. Axioms for cyclic interleaving

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td></td>
</tr>
<tr>
<td>(\left\llbracket \left\llbracket \left\llbracket S \right\rrbracket \right\rrbracket \right\rrbracket \cdot \alpha \right\rrbracket = \left\llbracket \left\llbracket \left\llbracket \alpha \right\rrbracket \right\rrbracket \right\rrbracket</td>
<td>CSI2</td>
</tr>
<tr>
<td>(\left\llbracket \left\llbracket D \right\rrbracket \right\rrbracket \cdot \alpha \right\rrbracket = DS_2 \left\llbracket \left\llbracket \alpha \right\rrbracket \right\rrbracket</td>
<td>CSI3</td>
</tr>
<tr>
<td>(\left\llbracket \left\llbracket x \triangleright a \triangleright y \right\rrbracket \right\rrbracket \cdot \alpha \right\rrbracket = \left\llbracket \left\llbracket \alpha \cdot \left\llbracket x \right\rrbracket \right\rrbracket \right\rrbracket \triangleright a \triangleright \left\llbracket \left\llbracket \alpha \cdot \left\llbracket y \right\rrbracket \right\rrbracket \right\rrbracket</td>
<td>CSI4</td>
</tr>
</tbody>
</table>

Table 6. Axioms for deadlock at termination

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(SD(S) = D)</td>
<td>S2D1</td>
</tr>
<tr>
<td>(SD(D) = D)</td>
<td>S2D2</td>
</tr>
<tr>
<td>(SD(x \triangleright a \triangleright y) = S_2D(x) \leq a \triangleright SD(y))</td>
<td>S2D3</td>
</tr>
</tbody>
</table>

Let \(H_i = (M_i, B_i, S_i, I_i, A_i, \llbracket \cdot \rrbracket_i)\), for \(i = 1, 2\), be Maurer machines with for all \(x \in M_1 \cap M_2\) either \(\forall I_1 \in I_1 \cdot x \notin OR(I_1)\) or \(\forall I_2 \in I_2 \cdot x \notin OR(I_2)\), and \(A_1 \cap A_2 = \emptyset\). Then the parallel composition of \(H_1\) and \(H_2\), written \(H_1 || H_2\), is the unique Maurer machine \((M, B, S, I, A, \llbracket \cdot \rrbracket)\) such that

- \(M = M_1 \cup M_2\),
- \(B = B_1 \cup B_2\),
- \(S = \{S : M \rightarrow B \mid S \in S_1 \land S \in S_2\}\),
- \(I = I_1 \cup I_2\),
- \(A = A_1 \cup A_2\),
- \(\llbracket a \rrbracket = \llbracket a \rrbracket_1\) if \(a \in A_1\),
- \(\llbracket a \rrbracket = \llbracket a \rrbracket_2\) if \(a \in A_2\).

Notice that the parallel composition of two Maurer machines is defined only if each common memory element is read-only for at least one of the Maurer machines. It is usual that the common memory elements do duty for communication between the parallel Maurer machines.

A thread vector is a sequence of threads. Strategic interleaving operators turn a thread vector of arbitrary length into a single thread. Several kinds of strategic interleaving have been elaborated in earlier work, see e.g. [5]. In this section, we only cover the simplest interleaving strategy, namely cyclic interleaving. The strategic interleaving operator for cyclic interleaving is denoted by \(||(\cdot)\)\).

The axioms for cyclic interleaving are given in Table 5. In CSI3, the auxiliary deadlock at termination operator \(SD(\cdot)\) is used. It turns termination into deadlock. Its axioms appear in Table 6.

The structural operational semantics of BTA extended with recursion, projection and cyclic interleaving is described by the transition rules given in Tables 2 and 7. Here \(\langle x, \rho \rangle \not\rightarrow\) stands for the set of all conditions \(\rightarrow \langle x, \rho \rangle \not\rightarrow (p', \rho')\) where \(p'\) is a closed term of this extension, \(\rho' \in \mathcal{E}, a \in A\).

Bisimulation equivalence is also a congruence with respect to the cyclic interleaving operator and the deadlock at termination operator. This follows imme-
such that for all $\text{smr} \not\in I$.

We also introduce a Maurer machine with a memory consisting of a thread location register ($\text{tlr}$), a basic action register ($\text{bar}$), a reply register ($\text{rr}$), and a stop mode register ($\text{smr}$), and the instruction set with a halt instruction ($I_{\text{halt}}$) and an execute stored basic action instruction ($I_{\text{esba}}$). Moreover, we replace the basic actions of the original Maurer machine by basic actions with which the extra instructions are associated. The resulting Maurer machines are called RST-enhancements. RST stands for remote stored thread.

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

Let $H = (M, B, S, I, A, [], \text{[ ]})$ be a Maurer machine with $M_i \not\subseteq M$, $\text{tlr}, \text{bar}, \text{rr}, \text{rrr}, \text{smr} \not\subseteq M$, $I_{\text{halt}} \not\subseteq I$, $I_{\text{fetch}} \not\subseteq I$ for all $r \in B$, $I_{\text{esba}} \not\subseteq I$, $\text{halt} \not\subseteq A$, $\text{fetchr} \not\subseteq A$ for all $r \in B$ and $\text{esba} \not\subseteq A$, and let $(I_a, m_a) = [a]$ for all $a \in A$. Then the RST-enhancement of $H$ is the Maurer machine $H' = (M', B', S', I', A', [], \text{[ ]})$ such that

\[ M' = M \cup \{\text{bar}, \text{rr}, \text{rrr}, \text{smr}\}, \]
\[ B' = B \cup A \cup B, \]

\text{immediately from the fact that the transition rules from Tables 2 and 7 constitute a complete transition system specification in relaxed panth format (see e.g. [12]).}

\section{Dealing with Finite-State Threads of Arbitrary Size}

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

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

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

Let $H = (M, B, S, I, A, [], \text{[ ]})$ be a Maurer machine with $M_i \not\subseteq M$, $\text{tlr}, \text{bar}, \text{rr}, \text{rrr}, \text{smr} \not\subseteq M$, $I_{\text{halt}} \not\subseteq I$, $I_{\text{fetch}} \not\subseteq I$ for all $r \in B$, $I_{\text{esba}} \not\subseteq I$, $\text{halt} \not\subseteq A$, $\text{fetchr} \not\subseteq A$ for all $r \in B$ and $\text{esba} \not\subseteq A$, and let $(I_a, m_a) = [a]$ for all $a \in A$. Then the RST-enhancement of $H$ is the Maurer machine $H' = (M', B', S', I', A', [], \text{[ ]})$ such that

\[ M' = M \cup \{\text{bar}, \text{rr}, \text{rrr}, \text{smr}\}, \]
\[ B' = B \cup A \cup B, \]
Here, \( I_{\text{halt}} \) is the unique function from \( S' \) to \( S' \) such that for all \( S' \in S' \):

\[
I_{\text{halt}}(S') \upharpoonright M = S' \upharpoonright M ,
\]

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

\[
I_{\text{halt}}(S')(\text{rr}) = S'(\text{smr}) ,
\]

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

\[
I_{\text{halt}}(S')(\text{smr}) = S'(\text{smr}) ;
\]

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

\[
I_{\text{exsba}}(S') \upharpoonright M = I_{S'(\text{bar})(S' \upharpoonright M)} \text{ if } S'(\text{rrr}) = T ,
\]

\[
I_{\text{exsba}}(S') \upharpoonright M = S' \upharpoonright M \text{ if } S'(\text{rrr}) = F ,
\]

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

\[
I_{\text{exsba}}(S')(\text{rr}) = I_{S'(\text{bar})(S' \upharpoonright M)}(m_{S'(\text{bar})}) \text{ if } S'(\text{rrr}) = T ,
\]

\[
I_{\text{exsba}}(S')(\text{rr}) = S'(\text{rr}) \text{ if } S'(\text{rrr}) = F ,
\]

\[
I_{\text{exsba}}(S')(\text{rrr}) = S'(\text{rrr}) ,
\]

\[
I_{\text{exsba}}(S')(\text{smr}) = S'(\text{smr}) .
\]

Moreover, the remote machine for stored threads is the Maurer machine \( H'' = (M'', B'', S'', I'', A'', \llbracket \cdot \rrbracket'') \) such that

\[
M'' = M_t \cup \{\text{trl, bar, rr, rrr, smr}\} ,
\]

\[
B'' = B_t \cup A_t \cup \{-1\} \cup A \cup B ,
\]

\[
S'' = \{ S'' : M'' \to B'' \mid M_t \in S_t \land S''(\text{trl}) \in A_t \cup \{-1\} \land S''(\text{bar}) \in A \land S''(rr) \in B \land S''(rrr) \in B \land S''(\text{smr}) \in B \} ,
\]

\[
I'' = \{ I_{\text{fetch}} \} ,
\]

\[
A'' = \{ \text{fetch} \} ,
\]

\[
\llbracket \text{fetch} \rrbracket'' = (I_{\text{fetch}}, \text{rrr}) .
\]

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

\[
I_{\text{fetch}}(S'') \upharpoonright M = S'' \upharpoonright M_t ,
\]

\[
I_{\text{fetch}}(S'')(\text{trl}) = \text{ntla}(S'', r) ,
\]

\[
\text{smr} = \text{exsba} .
\]
where \( r = S''(rr) \), and where \( \text{ntla} : S'' \times B \rightarrow A_c \) is defined as in the definition of an ST-4I-enhancement.

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

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

\[
\begin{align*}
CT' & = CT' \preceq \text{exsba} \triangleright (S \preceq \text{halt} \triangleright D) , \\
CT'' & = CT'' \preceq \text{fetch} \triangleright S .
\end{align*}
\]

\( CT' \) and \( CT'' \) are control threads for RST-enhancements of Maurer machines and remote machines for stored threads, respectively. Applying thread \( p \) to a state of Maurer machine \( H \) has the same effect as applying the cyclic interleaving of control threads \( CT' \) and \( CT'' \), starting with \( CT'' \), to the corresponding state of the parallel composition of the RST-enhancement of \( H \) and the remote machine for stored threads in which the thread memory contains the stored graph representation of \( p \).

**Theorem 10 (RST-enhancement).** Let \( H' = (M', B', S', I', A', [\ldots]) \) be the RST-enhancement of \( H = (M, B, S, I, A, [\ldots]) \), let \( H'' \) be the remote machine for stored threads, let \( p \in T_{\text{proc}}(A) \) be such that \( \text{size}(p) \leq \text{size}(M_A) \), let \( S^* \) be the set of states of \( H' \parallel H'' \), and let \( S_0' \in S^* \) be such that \( S_0' \parallel [M_0, \text{size}(p) - 1] = s_0(p) \), \( S_0'(\text{tir}) = -1, S_0'(rr) = T \). Then \( p \cdot_H (S_0' \parallel M) = ((CT'') \preceq (CT')) \cdot_H (\parallel H' \parallel S_0') \parallel M \).

**Proof.** Firstly, \( ((CT'') \preceq (CT')) \) is the solution of the guarded recursive specification over BTA that consists of the following equation:

\[
CT^* = (CT' \preceq \text{exsba} \triangleright CT^*) \preceq \text{fetch} \triangleright CT' ,
\]

where \( CT^{**} \) abbreviates \( (CT'') \preceq (S \preceq \text{halt} \triangleright D) \).
Secondly, \( H' \parallel H'' \) is the Maurer machine \( H = (M^*, B^*, S^*, I^*, A^*, [I]^*) \) such that

\[
\begin{align*}
M^* &= M \cup M_t \cup \{ \text{tlr, bar, rr, rrr, smr} \}, \\
B^* &= B \cup B_t \cup A_t \cup \{-1\} \cup A \cup B, \\
S^* &= \{ S^* : M^* \rightarrow B^* \mid \}
\]

\[
S^* \mid M \in S \land S^* \mid M_t \in S_t \land S^* \mid (\text{tlr}) \in A_t \cup \{-1\} \land \\
S^* \mid (\text{bar}) \in A \land S^* \mid (\text{rrr}) \in B \land S^* \mid (\text{smr}) \in B \}, \\
I^* &= \{ I^* : S^* \rightarrow S^* \mid \exists I \in I \land \forall S^* \in S^* \} \\
I^*(S^*) \mid M = I(S^* \mid M) \land I^*(S^*) \mid (M^* \setminus M) = S^* \mid (M^* \setminus M) \}
\]

\[
\cup \{ I_{\text{halt}}, I_{\text{fetch}}, I_{\text{exsba}} \}, \\
A^* &= \{ \text{halt, fetch, exsba} \}, \\
[I_{\text{halt}}]^* &= (I_{\text{halt}}, rr), \\
[I_{\text{fetch}}]^* &= (I_{\text{fetch}}, rrr), \\
[I_{\text{exsba}}]^* &= (I_{\text{exsba}}, rrr).
\]

Here, \( I_{\text{halt}}, I_{\text{fetch}} \) and \( I_{\text{exsba}} \) are the extensions of the instructions \( I_{\text{halt}}, I_{\text{fetch}} \) and \( I_{\text{exsba}} \) of \( H' \) and \( H'' \) to \( S^* \) such that \( I_{\text{halt}}(S^*) \mid (M^* \setminus M') = S^* \mid (M^* \setminus M') \), \\
\( I_{\text{fetch}}(S^*) \mid (M^* \setminus M') = S^* \mid (M^* \setminus M'') \) and \( I_{\text{exsba}}(S^*) \mid (M^* \setminus M') = S^* \mid (M^* \setminus M') \).

The remainder of the proof follows the same line as the proof of Theorem 8.

\[ \square \]

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

\section{Maurer Machines and Turing Machines}

Turing machines were proposed almost 70 years ago by Turing in [13]. In this section, we relate Maurer machines and Turing machines. First of all, we show how Turing machines can be simulated by means of Maurer machines.

Assume that a fixed but arbitrary countably infinite set \( M_{\text{tape}} \) and a fixed but arbitrary bijection \( m_{\text{tape}} : N \rightarrow M_{\text{tape}} \) have been given. \( M_{\text{tape}} \) is called the \textit{tape}. Let \( n \in N \). Then we write \( M_{\text{tape}}[n] \) for \( m_{\text{tape}}(n) \).

The tape is an infinite memory of which the elements can be addressed by means of elements of \( N \). We write \( A_{\text{tape}} \) for \( N \). The elements of the tape contain 0, 1 or \( \square \) (blank). We write \( B_{\text{tape}} \) for the set \( \{0, 1, \square\} \), and 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 \( n \in A_{\text{tape}} \) such that for all \( m \in A_{\text{tape}} \) with \( m \geq n \), \( S_{\text{tape}}(M_{\text{tape}}[m]) = \square \).

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

\[
M = M_{\text{tape}} \cup \{\text{head}, rr\}, \\
B = B_{\text{tape}} \cup A_{\text{tape}} \cup \mathbb{B}, \\
S = \{S : M \rightarrow B | S \upharpoonright M_{\text{tape}} = S_{\text{tape}} \land S(\text{head}) \in A_{\text{tape}} \land S(rr) \in \mathbb{B}\}, \\
I = \{I_{\text{test}\_s}, I_{\text{write}\_s} | s \in B_{\text{tape}}\} \cup \{I_{\text{mover}}, I_{\text{move}}\}, \\
A = \{\text{test}:s, \text{write}:s | s \in B_{\text{tape}}\} \cup \{\text{mover}, \text{move}\}, \\
[a] = (I_a, rr) \text{ for all } a \in A.
\]

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

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

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 \(n \in A_{\text{tape}}\):

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

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

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

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

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

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 \(\{X_0|\{X_0 = t_0, \ldots, X_n = t_n\}\} \in T_{\text{finrec}}\), 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 \(t, t', t''\) of the form \(\text{write}:s \circ X, \text{mover} \circ X, X \leq \text{move} \geq D\) or \(S\) \((s \in B_{\text{tape}}, X \in \{X_0, \ldots, X_n\})\).
Clearly, for each Turing machine, there is a Turing thread \( p \in T_{\text{rec}}(A_{TM}) \) such that for all \( S \in S_{TM} \), \( p \circ_{TM} S \) can simulate the computations of that Turing machine from the initial tape contents \( S \uparrow M_{\text{tape}} \) and initial head position \( S(\text{head}) \).

Looking at the instructions used in the simulation of Turing machines by means of a Maurer machine, we observe that the test instructions \( I_{\text{test}:s:n} \) have an infinite input region and a finite output region and that the write instructions \( I_{\text{write}:s:n} \) have a finite input region and an infinite output region. It is not difficult to see that these infinite regions are essential for the simulation of many Turing machines. 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.

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.

It is assumed that \( \text{test}:s:n, \text{write}:s:n \in A \) for all \( s \in B_{\text{tape}} \) and \( n \in A_{\text{tape}} \). \( TM' \) is the Maurer machine \( (M', B', S', T', A', [\cdot]) \) such that

\[
M' = M_{\text{tape}} \cup \{\text{rr}\},
\]

\[
B' = B_{\text{tape}} \cup B,
\]

\[
S' = \{S': M' \rightarrow B' | S' \uparrow M_{\text{tape}} = S_{\text{tape}} \wedge S'(\text{rr}) \in B\},
\]

\[
T' = \{I_{\text{test}:s:n}, I_{\text{write}:s:n} | s \in B_{\text{tape}} \wedge n \in A_{\text{tape}}\},
\]

\[
A' = \{\text{test}:s:n, \text{write}:s:n | s \in B_{\text{tape}} \wedge n \in A_{\text{tape}}\},
\]

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

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

\[
\begin{align*}
I_{\text{test}:s:n}(S')(S' \uparrow M_{\text{tape}}) &= S' \uparrow M_{\text{tape}}, \\
I_{\text{test}:s:n}(S')(\text{rr}) &= T \quad \text{if } S'(M_{\text{tape}}[n]) = s, \\
I_{\text{test}:s:n}(S')(\text{rr}) &= F \quad \text{if } S'(M_{\text{tape}}[n]) \neq s;
\end{align*}
\]

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

\[
\begin{align*}
I_{\text{write}:s:n}(S')(M_{\text{tape}}[n]) &= s, \\
I_{\text{write}:s:n}(S')(M_{\text{tape}}[m]) &= S'(M_{\text{tape}}[m]) \quad \text{if } n \neq m, \\
I_{\text{write}:s:n}(S')(\text{rr}) &= T.
\end{align*}
\]

\[\text{We consider Turing machines with one-way infinite tape, which are as powerful as Turing machines with two-way infinite tape. We interpret the usual remark “no left move is permitted when the read-write head is at the [left] boundary” (see e.g. [9]) as “a left move leads to inaction when the read-write head is at the left boundary.”}\]
We write $S_{TM'}$ and $A_{TM'}$ for the set of states of $TM'$ and the set of basic actions of $TM'$, respectively.

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 subterms of a term $t \in \mathbb{T}_{\text{finrec}}$ 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 $\mathcal{H}_{X_i}(t, l) \leq \text{test}: s \geq t' \in T_0$ 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 $\mathcal{H}_{X_i}(t, l), t \leq \text{mover} \geq D \in T_0$ and $l \geq 0$, then $\mathcal{H}_{X_i}(t \leq \text{mover} \geq D, l - 1)$.

We write $X_0 \xrightarrow{k,E} X_i$ to indicate that there exist a $t \in \mathbb{T}_{\text{finrec}}$ 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 $X_0 \xrightarrow{k,E} X_i$, then $X_{ik} = t_{ik} \in \psi(E)$, where $t_{ik}$ is obtained from $t_i$ by applying the following replacement rules:
  - test:$s$ is replaced by test:$s:k$;
  - write:$s \circ X_j$ is replaced by write:$s:k \circ X_{jk}$;
  - mover$\circ X_j$ is replaced by $X_{jl}$, where $l = k + 1$;
  - if $k \neq 0$, then $X_j \leq \text{mover} \geq D$ is replaced by $X_{jl}$, where $l = k - 1$;
  - if $k = 0$, then $X_j \leq \text{mover} \geq D$ is replaced by $D$.

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

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

**Theorem 11 (Role of head position in Turing machines).** Let $p$ be a Turing thread, and let $S_0 \in S_{TM}$ be such that $S_0(\text{head}) = 0$. Then $(p \bullet_{TM} S_0) \upharpoonright (M_{\text{tape}} \cup \{\tau\}) = \psi(p) \bullet_{TM'} (S_0 \upharpoonright (M_{\text{tape}} \cup \{\tau\}))$.

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

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

where $n = S(\text{head})$. 26
Let \((p_n, S_n)\) be the \(n+1\)-th element in the full path of \(p \cdot TM S_0\) of which the first element equals \(S\), \(D\) or \(q \triangleright test:0 \triangleright r\) for some \(q, r \in T_{finrec}\), and let \((p'_n, S'_n)\) be the \(n+1\)-th element in the full path of \(\psi(p) \cdot TM' (S_0 \mid (M_{tape} \cup \{rr\}))\) of which the first element equals \(S\), \(D\) or \(q' \triangleright test:0:k \triangleright r'\) for some \(k \in A_{tape}\) and \(q', r' \in T_{finrec}\).

Then, using the above equations, it is straightforward to prove by induction on \(n\) that:

- \(p_{n+1} = q \triangleleft test:0 \triangleright r\) and \(S_{n+1}((\text{head})) = k\) iff \(p'_{n+1} = q' \triangleleft test:0:k \triangleright r'\) with \(q'\) and \(r'\) obtained from \(q\) and \(r\) by applying the replacement rules given in the definition of \(\psi\) above;
- \(S_{n+1} \upharpoonright (M_{tape} \cup \{rr\}) = S'_{n+1}\).

(for \(n+1 < |p \cdot TM S_0|\) if \(p \cdot TM S_0\) is convergent). From this, the theorem follows immediately. \(\Box\)

Consider the Turing thread \(\langle X_0 \mid E \rangle\), where

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

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

Clearly, \(X_0 \xrightarrow{E} X_0\) for each \(k \in \mathbb{N}\). Consequently, in this case \(\psi(E)\) is a countably infinite set. We conclude the following concerning Turing machines:

- 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.

13 Stored Threads and Programs

In this section, we discuss the connection between stored threads and programs. First, we review the program notation PGLD [4], which is close to existing assembly languages.

In PGLD, it is assumed that there is a fixed but arbitrary set of basic instructions \(I\). PGLD has the following primitive instructions:

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

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

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

We write \(\mathcal{P}_{\text{PGLD}}\) for the set of all PGLD programs.

The behaviour of a PGLD program, as defined in [4], is a thread which is
definable by a finite guarded recursive specification over BTA. It is easy to see
that each finite guarded recursive specification over BTA can be translated to
a PGLD program of which the behaviour is the solution of the finite guarded
recursive specification concerned (cf. Section 5 of [3]).

Next, we consider the stored threads from Section 6 again. We write \(\mathcal{ST}\)
for \(\{s_p : p \in T_{\text{intrec}} \land \text{size}(p) \leq \text{size}(M)\}\). We define a translation function
\(\text{pgld} : \mathcal{ST} \rightarrow \mathcal{P}_{\text{PGLD}}\) for stored threads. For all \(T \in \mathcal{ST}\), \(\text{pgld}(T) = \text{pgld}'(T, 0)\),
where \(\text{pgld}' : \mathcal{ST} \times \mathbb{N} \rightarrow \mathcal{P}_{\text{PGLD}}\) is recursively defined as follows:

\[
\begin{align*}
\text{pgld}'(T, n) &= \text{pgld}''(T, n) & \text{if } T(M_1[n+1]) \not\in \text{dom}(T), \\
\text{pgld}'(T, n) &= \text{pgld}''(T, n); \text{pgld}'(T, n+1) & \text{if } T(M_1[n+1]) \in \text{dom}(T),
\end{align*}
\]

where \(\text{pgld}'' : \mathcal{ST} \times \mathbb{N} \rightarrow \mathcal{P}_{\text{PGLD}}\) is defined as follows:

\[
\begin{align*}
\text{pgld}''(T, n) &= +a; ##3n' + 1; ##3n'' + 1 & \text{if } T(M_1[n]) = (n', a, n'') , \\
\text{pgld}''(T, n) &= ##0; ##0; ##0 & \text{if } T(M_1[n]) = S , \\
\text{pgld}''(T, n) &= ##3n + 1; ##3n + 2; ##3n + 3 & \text{if } T(M_1[n]) = D .
\end{align*}
\]

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

It can be shown that, for all \(p \in T_{\text{intrec}}\) with \(\text{size}(p) \leq \text{size}(M)\), the behaviour
of \(\text{pgld}(s_p(p))\) is the thread described by \(p\). The function \(\text{pgld}\) shows that there is
hardly a difference between the stored thread \(s_p(p)\) and the PGLD program
\(\text{pgld}(s_p(p))\) extracted from it: \(s_p(p)\) can also be viewed as a stored representation
of \(\text{pgld}(s_p(p))\) with three primitive instruction to a memory element. However, it is
likely that \(\text{pgld}(s_p(p))\) contains needless jump instructions. For example, what
can be achieved by a positive test instruction \(+a\) followed by two identical jump
instructions can also be achieved by a void basic instruction \(a\). In other words,
PGLD permits a more efficient representation of threads than the one obtained
by way of \(s_p\) and \(\text{pgld}\).
In view of the above, a thorough investigation into stored PGLD programs and their execution looks to be worthwhile. Because the behaviour of a PGLD program is a thread which is definable by a finite guarded recursive specification over BTA, it will be easy to define a counterpart of the apply operator from Section 4 for PGLD programs. It will be trivial to make precise how to represent PGLD programs in the memory of a Maurer computer. It will be straightforward to define counterparts of ST-4I-enhancements, ST-1I-enhancements and RST-enhancements for PGLD programs. It will be relatively involved to prove counterparts of Theorems 8, 9 and 10. Therefore, we leave these matters as a whole for future work.

An interesting feature of PGLD is that PGLD programs are close to terms of PGA (Program Algebra); and a mapping has been defined by which they can be turned into terms of PGA (see e.g. [4]). Using the axioms of PGA, programs can be simplified algebraically. For example, chained jumps can be removed and thus the size of the program can be reduced.

14 Conclusions

We have developed a theory which covers basic issues concerning stored threads and their execution. Coverage of more advanced issues relevant to the design of processor architectures, primarily pipelining, is now within reach. That would enable us to formalize recent ideas intended to speed up processors, such as micro-threading [8], and to prove or disprove anticipated speed-up results.

We are not aware of other theoretical work on stored threads and their execution that builds on a model for computers as close to real computers as Maurer’s model. It happens that thread algebra fits in with Maurer computers very well. This comes as a mild surprise because Maurer computers were not at all within our sight at the time thread algebra was developed.

References