On the operating unit size of load/store architectures

Bergstra, J.A.; Middelburg, C.A.

DOI
10.1017/S0960129509990314
Publication date
2010
Document Version
Final published version
Published in
Mathematical Structures in Computer Science

Math. Struct. in Comp. Science (2010), *vol.* 20, *pp.* 395–417.
doi:10.1017/S0960129509990314 First published online 4 February 2010

On the operating unit size of load/store architectures[†]

J. A. BERGSTRA and C. A. MIDDELBURG

Informatics Institute, University of Amsterdam, Science Park 904, 1098 XH Amsterdam, the Netherlands
Email: {J.A.Bergstra,C.A.Middelburg}@uva.nl

Received 3 August 2007; revised 13 January 2009

We introduce a strict version of the concept of a load/store instruction set architecture in the setting of Maurer machines. We take the view that transformations on the states of a Maurer machine are achieved by applying threads as considered in thread algebra to the Maurer machine. We study how the transformations on the states of the main memory of a strict load/store instruction set architecture that can be achieved by applying threads depend on the operating unit size, the cardinality of the instruction set and the maximal number of states of the threads.

1. Introduction

In Bergstra and Middelburg (2007), we introduced Maurer machines, which are based on the model for computers proposed in Maurer (1966), and extended basic thread algebra, which was introduced in Bergstra and Loots (2002) under the name of basic polarised process algebra, with operators for applying threads to Maurer machines. Threads can be looked upon as the behaviours of deterministic sequential programs as run on a machine. By applying threads to a Maurer machine, transformations on the states of the Maurer machine are achieved. In Bergstra and Middelburg (2008), we proposed a strict version of the concept of a load/store instruction set architecture for theoretical work relevant to the design of micro-architectures (architectures of micro-processors). We described the concept in the setting of Maurer machines. The idea underlying it is that there is a main memory whose elements contain data, an operating unit with a small internal memory through which data can be manipulated, and an interface between the main memory and the operating unit for data transfer between them. The bit size of the operating unit memory of a load/store instruction set architecture is called its operating unit size.

In this paper, we study how the transformations on the states of the main memory of a strict load/store instruction set architecture that can be achieved by applying threads to it depend on the operating unit size, the cardinality of the instruction set

[†] This research was carried out in part in the framework of the GLANCE-project MICROGRIDS, which is funded by the Netherlands Organisation for Scientific Research (NWO).

and the maximal number of states of the threads. The motivation for this work is our assumption that load/store instruction set architectures impose restrictions on the expressiveness of computers, evidence for which is produced in the paper. To assist in the presentation of certain results, we introduce the concept of a thread powered function class. The idea underlying this concept is that the transformations on the main memory of strict load/store instruction set architectures achievable by applying threads to them are primarily determined by the address width, the word length, the operating unit size and the cardinality of their instruction set, and the number of states of the threads that can be applied to them.

We choose to use Maurer machines and basic thread algebra to study issues relevant to the design of instruction set architectures. Maurer machines are based on the view that a computer has a memory, the contents of all memory elements make up the state of the computer, the computer processes instructions and the processing of an instruction amounts to performing an operation on the state of the computer that results in changes of the contents of certain memory elements. The design of instruction set architectures must deal with these aspects of real computers. Turing machines and the other kinds of machines used in theoretical computer science (see, for example, Hopcroft *et al.* (2001)) abstract from these aspects of real computers. Basic thread algebra is a form of process algebra. Well-known process algebras, such as ACP (Baeten and Weijland 1990), CCS (Milner 1989) and CSP (Hoare 1985), are too general for our purposes, *viz.* modelling deterministic sequential processes that interact with a machine. Basic thread algebra was designed as an algebra of processes with this end in mind. We showed in Bergstra and Middelburg (2006) that the processes considered in basic thread algebra can be viewed as processes that are definable over ACP. However, modelling and analysing them is rather difficult using such a general process algebra.

The structure of this paper is as follows. First, we review basic thread algebra (Section 2), Maurer machines (Section 3) and the operators for applying threads to Maurer machines (Section 4). In Section 5, we introduce the concept of a strict load/store Maurer instruction set architecture. In Section 6, we study the consequences of reducing the operating unit size of a strict load/store Maurer instruction set architecture. In Section 7, we give conditions under which all possible transformations on the states of the main memory of a strict load/store Maurer ISA with a certain address width and word length can be achieved by applying a thread to such a strict load/store Maurer ISA. In Section 8, we give a condition under which not all possible transformations can be achieved. Finally, Section 9 presents some concluding remarks.

2. Basic thread algebra

In this section, we review BTA (Basic Thread Algebra). This is a form of process algebra, which was first presented in Bergstra and Loots (2002) under the name BPPA (Basic Polarised Process Algebra), that is tailored to the description of the behaviour of deterministic sequential programs under execution. These behaviours are called *threads*.

$\langle X\|E\rangle = \langle t_X\|E\rangle$	if $X = t_X \in E$	RDP
$E \Rightarrow X = \langle X\|E\rangle$	if $X \in V(E)$	RSP

Table 1. *Axioms for guarded recursion*

In BTA, it is assumed that there is a fixed but arbitrary set of *basic actions* $\mathscr{A}$. BTA has the following constants and operators:

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

We use infix notation for postconditional composition. We introduce *action prefixing* as an abbreviation: $a \circ p$, where p is a term of BTA, abbreviates $p \trianglelefteq a \trianglerighteq p$.

The intuition is that each basic action performed by a thread is taken as a command to be processed by the execution environment of the thread. The processing of a command may involve a change of state of the execution environment. At completion of the processing of the command, the execution environment produces a reply. This reply is either T or F and is returned to the thread concerned. Let p and q be closed terms of BTA. Then $p \trianglelefteq a \trianglerighteq q$ will perform action a, and after that proceed as p if the processing of a leads to the reply T (called a positive reply) and proceed as q if the processing of a leads to the reply F (called a negative reply).

Each closed term of BTA denotes a finite thread, that is, a thread where the length of the sequences of actions that it can perform is bounded. Guarded recursive specifications can give rise to infinite threads.

A *guarded recursive specification* over BTA is a set of recursion equations $E = \{X = t_X \mid X \in V\}$, where V is a set of variables and each t_X is a term of the form D, S or $t \trianglelefteq a \trianglerighteq t'$ with t and t' terms of BTA that contain only 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. 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 Bergstra and Bethke (2003).

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 to the constants of BTA a constant standing for the unique solution of E for X, which we denote by $\langle X|E\rangle$. We also add the axioms for guarded recursion given in Table 1 to BTA, where we write $\langle t_X|E\rangle$ for t_X with, for all $Y \in V(E)$, all occurrences of Y in t_X replaced by $\langle Y|E\rangle$. In Table 1, X, t_X and E stand for an arbitrary variable, an arbitrary term of BTA and an arbitrary guarded recursive specification, respectively. Side conditions are added to restrict the variables, terms and guarded recursive specifications for which X, t_X and E stand. The additional axioms for guarded recursion are known as the recursive definition principle (RDP) and the recursive specification principle (RSP). The equations $\langle X|E\rangle = \langle t_X|E\rangle$ for a fixed E express the fact that the constants $\langle X|E\rangle$ make up a solution of E. The conditional equations $E \Rightarrow X = \langle X|E\rangle$ express the fact that this solution is the only one.

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

Table 2. *Approximation induction principle*

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

Table 3. *Axioms for projection operators*

We will write BTA+REC for BTA extended with the constants for solutions of guarded recursive specifications and axioms RDP and RSP.

Closed terms of BTA+REC that denote the same infinite thread cannot always be proved to be equal using the axioms of BTA+REC. We introduce the approximation induction principle to remedy this. The approximation induction principle, AIP, is based on the view that two threads are identical if their approximations up to any finite depth are identical. The approximation up to depth n of a thread is obtained by cutting it off after performing a sequence of actions of length n.

AIP is the infinitary conditional equation given in Table 2. Here, following Bergstra and Bethke (2003), the approximation of depth n is given in terms of a unary *projection* operator $\pi_n(\_)$. The axioms for the projection operators are given in Table 3, where a stands for an arbitrary member of $\mathscr{A}$.

From now on, we will write $\mathscr{E}_{\mathsf{fin}}(A)$, where $A \subseteq \mathscr{A}$, for the set of all finite guarded recursive specifications over BTA that only contain postconditional operators $\_ \trianglelefteq a \trianglerighteq \_$ for which $a \in A$. Moreover, we write $\mathscr{T}_{\mathsf{finrec}}(A)$, where $A \subseteq \mathscr{A}$, for the set of all closed terms of BTA+REC that only contain postconditional operators $\_ \trianglelefteq a \trianglerighteq \_$ for which $a \in A$ and only contain constants $\langle X|E\rangle$ for which $E \in \mathscr{E}_{\mathsf{fin}}(A)$.

A *linear recursive specification* over BTA is a guarded recursive specification $E = \{X = t_X \mid X \in V\}$, where each t_X is a term of the form D, S or $Y \trianglelefteq a \trianglerighteq Z$ with $Y, Z \in V$. For each closed term $p \in \mathscr{T}_{\mathsf{finrec}}(A)$, there exist a linear recursive specification $E \in \mathscr{E}_{\mathsf{fin}}(A)$ and a variable $X \in V(E)$ such that $p = \langle X|E\rangle$ is derivable from the axioms of BTA+REC.

From now on, we will write $\mathscr{E}^{\mathsf{lin}}_{\mathsf{fin}}(A)$, where $A \subseteq \mathscr{A}$, for the set of all linear recursive specifications from $\mathscr{E}_{\mathsf{fin}}(A)$.

In the following, the interpretations of the constants and operators of BTA+REC in models of BTA+REC will be denoted by the constants and operators themselves. Let $\mathscr{M}$ be some model of BTA+REC, and p be an element from the domain of $\mathscr{M}$. Then the set of *states* or *residual threads* of p, written $Res(p)$, is defined inductively as follows:

— $p \in Res(p)$;
— if $q \trianglelefteq a \trianglerighteq r \in Res(p)$, then $q \in Res(p)$ and $r \in Res(p)$.

We are only interested in models of BTA+REC in which $card(Res(\langle X|E\rangle)) \leqslant card(E)$ for all finite linear recursive specifications E, such as the projective limit model of BTA presented in Bergstra and Bethke (2003).

3. Maurer machines

In this section, we review the concept of a Maurer machine. This concept was first introduced in Bergstra and Middelburg (2007).

A *Maurer machine* H consists of the following components:

— a non-empty set M;
— a set B with $card(B) \geqslant 2$;
— a set $\mathscr{S}$ of functions $S : M \to B$;
— a set $\mathscr{O}$ of functions $O : \mathscr{S} \to \mathscr{S}$;
— a set $A \subseteq \mathscr{A}$;
— a function $[\![\_]\!] : A \to (\mathscr{O} \times M)$.

It also satisfies the following conditions:

— if $S_1, S_2 \in \mathscr{S}$, $M' \subseteq M$, and $S_3 : M \to B$ is such that $S_3(x) = S_1(x)$ if $x \in M'$ and $S_3(x) = S_2(x)$ if $x \notin M'$, then $S_3 \in \mathscr{S}$;
— if $S_1, S_2 \in \mathscr{S}$, then the set $\{x \in M \mid S_1(x) \neq S_2(x)\}$ is finite;
— if $S \in \mathscr{S}$, $a \in A$, and $[\![a]\!] = (O, m)$, then $S(m) \in \{\mathsf{T}, \mathsf{F}\}$.

M is called the *memory* of H, and B is called the *base set* of H. The members of $\mathscr{S}$ are called the *states* of H, the members of $\mathscr{O}$ are called the *operations* of H, the members of A are called the *basic actions* of H and $[\![\_]\!]$ is called the *basic action interpretation function* of H.

We write M_H, B_H, $\mathscr{S}_H$, $\mathscr{O}_H$, A_H and $[\![\_]\!]_H$, where $H = (M, B, \mathscr{S}, \mathscr{O}, A, [\![\_]\!])$ is a Maurer machine, for M, B, $\mathscr{S}$, $\mathscr{O}$, A and $[\![\_]\!]$, respectively.

A Maurer machine has much in common with a real computer. The memory of a Maurer machine consists of memory elements whose contents are elements from its base set. The term memory must not be interpreted too strictly. For example, register files and caches must be regarded as parts of the memory. The contents of all memory elements together make up a state of the Maurer machine. State changes are accomplished by performing its operations. Every state change amounts to a change in the contents of certain memory elements. The basic actions of a Maurer machine are the instructions that it is able to process. The processing of a basic action amounts to performing the operation associated with the basic action using the basic action interpretation function. At completion of the processing, the content of the memory element associated with the basic action by the basic action interpretation function is the reply produced by the Maurer machine. The term basic action originates from BTA.

The first condition on the states of a Maurer machine is a structural condition and the second is a finite variability condition. We will return to these conditions, which are met by any real computer, after we have introduced the input and output regions of an operation. The third condition on the states of a Maurer machine restricts the possible replies at completion of the processing of a basic action to T or F.

Maurer (1966) proposed a model for computers. We then introduced the term Maurer computer in Bergstra and Middelburg (2007) to denote a computer conforming to Maurer's definition. Separating out the set of basic actions and the basic action interpretation function from a Maurer machine leaves a Maurer computer. The set of

basic actions and the basic action interpretation function constitute the interface of a Maurer machine with its environment, which effectuates state changes by issuing basic actions.

The notions of the input and output regions of an operation, which originated in Maurer (1966), will be used in later sections.

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

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

$$OR(O)) = \{x \in M \mid \exists S \in \mathscr{S} \cdot S(x) \neq O(S)(x)\}^{\dagger}.$$

$OR(O)$ is the set of all memory elements that might possibly be affected by O; and $IR(O)$ is the set of all memory elements that might possibly affect elements of $OR(O)$ under O. For example, the input and output regions of an operation that adds the content of a given main memory cell, say X, to the content of a given register, say $R0$, are $\{X, R0\}$ and $\{R0\}$, respectively.

Let $H = (M, B, \mathscr{S}, \mathscr{O}, A, [\![\_]\!])$ be a Maurer machine. Let $S_1, S_2 \in \mathscr{S}$ and $O \in \mathscr{O}$. Then $S_1 \restriction IR(O) = S_2 \restriction IR(O)$ implies $O(S_1) \restriction OR(O) = O(S_2) \restriction OR(O)$[‡]. In other words, every operation transforms states that coincide on the input region of the operation to states that coincide on the output region of the operation. The second condition on the states of a Maurer machine is required for this fundamental property to hold. The first condition on the states of a Maurer machine could be relaxed somewhat.

More results relating to input regions and output regions are given in Maurer (1966). Recently, a revised and expanded version of Maurer (1966), which includes all the proofs, has appeared as Maurer (2006).

4. Applying threads to Maurer machines

In this section, we add a binary *apply* operator $\_ \bullet_H \_$ to BTA+REC for each Maurer machine H, and introduce a notion of computation in the resulting setting.

The apply operators associated with Maurer machines are related to the apply operators introduced in Bergstra and Ponse (2002). They allow threads to transform states of the associated Maurer machine by means of its operations. Such state transformations produce either a state of the associated Maurer machine or the *undefined state* $\uparrow$. It is assumed that $\uparrow$ is not a state of any Maurer machine. We extend function restriction to $\uparrow$ by stipulating that $\uparrow \restriction M = \uparrow$ for any set M. The first operand of the apply operator $\_ \bullet_H \_$ associated

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

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

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

Table 4. *Defining equations for apply operator*

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

Table 5. *Rule for divergence*

with Maurer machine $H = (M, B, \mathscr{S}, \mathscr{O}, A, [\![\_]\!])$ must be a term from $\mathscr{T}_{\mathsf{finrec}}(A)$, and its second argument must be a state from $\mathscr{S} \cup \{\uparrow\}$.

Let $H = (M, B, \mathscr{S}, \mathscr{O}, A, [\![\_]\!])$ be a Maurer machine, $p \in \mathscr{T}_{\mathsf{finrec}}(A)$ and $S \in \mathscr{S}$. Then $p \bullet_H S$ is the state that results if all basic actions performed by thread p are processed by the Maurer machine H from initial state S. The processing of a basic action a by H amounts to a state change according to the operation associated with a by $[\![\_]\!]$. In the resulting state, the reply produced by H is contained in the memory element associated with a by $[\![\_]\!]$. If p is S, there will be no state change; if p is D, the result is $\uparrow$.

Let $H = (M, B, \mathscr{S}, \mathscr{O}, A, [\![\_]\!])$ be a Maurer machine and $(O_a, m_a) = [\![a]\!]$ for all $a \in A$. Then the apply operator $\_ \bullet_H \_$ is defined by the equations given in Table 4 and the rule given in Table 5. In these tables, a stands for an arbitrary member of A, and S stands for an arbitrary member of $\mathscr{S}$.

Let $H = (M, B, \mathscr{S}, \mathscr{O}, A, [\![\_]\!])$ be a Maurer machine. Let $p \in \mathscr{T}_{\mathsf{finrec}}(A)$ and $S \in \mathscr{S}$. Then p *converges* from S on H if there exists an $n \in \mathbb{N}$ such that $\pi_n(p) \bullet_H S \neq \uparrow$. The rule from Table 5 can be read as follows: if x does not converge from S on H, then $x \bullet_H S$ equals $\uparrow$.

Before we introduce a notion of computation in the current setting, we need to introduce some auxiliary notions.

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

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

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

— a finite path $\langle (p_0, S_0), \ldots, (p_n, S_n) \rangle$ in $\_ \vdash_H \_$ such that there exists no $(p_{n+1}, S_{n+1}) \in \mathscr{T}_{\mathsf{finrec}}(A) \times \mathscr{S}$ with $(p_n, S_n) \vdash_H (p_{n+1}, S_{n+1})$;
— an infinite path $\langle (p_0, S_0), (p_1, S_1), \ldots \rangle$ in $\_ \vdash_H \_$.

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

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

5. Instruction set architectures

In this section, we introduce the concept of a strict load/store Maurer instruction set architecture. This concept, which was first introduced in Bergstra and Middelburg (2008), takes its name from the following features:

— it is described in the setting of Maurer machines;
— it is only concerned with load/store architectures; and
— the load/store architectures concerned are strict in some respects – this will be explained after we have completed the formalisation.

The concept of a strict load/store Maurer instruction set architecture, or a strict load/store Maurer ISA for short, is an approximation of the concept of a load/store instruction set architecture (see, for example, Hennessy and Patterson (2003)). It is focussed on instructions for data manipulation and data transfer. The transfer of program control is treated in a uniform way over different strict load/store Maurer ISAs by working at the abstraction level of threads. All that is left of the transfer of program control at this level is postconditional composition.

Each Maurer machine has a number of basic actions with which an operation is associated. From now on, in the context of Maurer machines that are strict load/store Maurer ISAs, such basic actions will be referred to loosely as basic instructions – the term basic action is uncommon when we are concerned with ISAs.

The idea underlying the concept of a strict load/store Maurer ISA is that there is a main memory whose elements contain data, an operating unit with a small internal memory through which data can be manipulated, and an interface between the main memory and the operating unit for data transfer between them. For the sake of simplicity, data is restricted to the natural numbers between 0 and some upper bound. Other types of data that one might want to support can always be represented by the natural numbers provided. Moreover, the data manipulation instructions offered by a strict load/store Maurer ISA are not restricted, and may include ones that are tailored to the manipulation of representations of other types of data. Therefore, we believe that nothing essential is lost by the restriction to natural numbers.

The concept of a strict load/store Maurer ISA is parametrised by:

— an address width aw;
— a word length wl;
— an operating unit size ous;
— a number $nrpl$ of pairs of data and address registers for load instructions;
— a number $nrps$ of pairs of data and address registers for store instructions;
— a set A_{dm} of basic instructions for data manipulation;

where $aw, ous \geqslant 0$, $wl, nrpl, nrps > 0$ and $A_{dm} \subseteq \mathscr{A}$.

The address width *aw* may be thought of as the number of bits used for the binary representation of addresses of data memory elements. The word length *wl* may be thought of as the number of bits used to represent data in data memory elements. The operating unit size *ous* may be thought of as the number of bits that the internal memory of the operating unit contains. The operating unit size is measured in bits because this allows us to establish results for which no assumptions about the internal structure of the operating unit are involved.

We assume that, for each $n \in \mathbb{N}$, a fixed but arbitrary countably infinite set $\mathsf{M}^n_{\mathsf{data}}$ and a fixed but arbitrary bijection $\mathsf{m}^n_{\mathsf{data}} : \mathbb{N} \to \mathsf{M}^n_{\mathsf{data}}$ are given. The members of $\mathsf{M}^n_{\mathsf{data}}$ are called *data memory elements*. The contents of data memory elements are taken as data. The data memory elements from $\mathsf{M}^n_{\mathsf{data}}$ can contain natural numbers in the interval $[0, 2^n - 1]$.

We assume that a fixed but arbitrary countably infinite set M_{ou} and a fixed but arbitrary bijection $\mathsf{m}_{\mathsf{ou}} : \mathbb{N} \to \mathsf{M}_{\mathsf{ou}}$ are given. The members of M_{ou} are called *operating unit memory elements*. They can contain natural numbers in the set $\{0, 1\}$, that is, bits. Usually, a part of the operating unit memory is partitioned into groups that data manipulation instructions can refer to.

We assume that, for each $n \in \mathbb{N}$, fixed but arbitrary countably infinite sets $\mathsf{M}^n_{\mathsf{ld}}$, $\mathsf{M}^n_{\mathsf{la}}$, $\mathsf{M}^n_{\mathsf{sd}}$ and $\mathsf{M}^n_{\mathsf{sa}}$ and fixed but arbitrary bijections $\mathsf{m}^n_{\mathsf{ld}} : \mathbb{N} \to \mathsf{M}^n_{\mathsf{ld}}$, $\mathsf{m}^n_{\mathsf{la}} : \mathbb{N} \to \mathsf{M}^n_{\mathsf{la}}$, $\mathsf{m}^n_{\mathsf{sd}} : \mathbb{N} \to \mathsf{M}^n_{\mathsf{sd}}$ and $\mathsf{m}^n_{\mathsf{sa}} : \mathbb{N} \to \mathsf{M}^n_{\mathsf{sa}}$ are given. The members of $\mathsf{M}^n_{\mathsf{ld}}$, $\mathsf{M}^n_{\mathsf{la}}$, $\mathsf{M}^n_{\mathsf{sd}}$ and $\mathsf{M}^n_{\mathsf{sa}}$ are called *load data registers*, *load address registers*, *store data registers* and *store address registers*, respectively. The contents of load data registers and store data registers are taken to be data, whereas the contents of load address registers and store address registers are taken to be addresses. The load data registers from $\mathsf{M}^n_{\mathsf{ld}}$, the load address registers from $\mathsf{M}^n_{\mathsf{la}}$, the store data registers from $\mathsf{M}^n_{\mathsf{sd}}$ and the store address registers from $\mathsf{M}^n_{\mathsf{sa}}$ can contain natural numbers in the interval $[0, 2^n - 1]$. The load and store registers are special memory elements designated for the transfer of data between the data memory and the operating unit memory.

We assume a single special memory element rr for passing on the replies resulting from the processing of basic instructions. This special memory element is called the *reply register*.

We assume that, for each $n, n' \in \mathbb{N}$, $\mathsf{M}^n_{\mathsf{data}}$, M_{ou}, $\mathsf{M}^n_{\mathsf{ld}}$, $\mathsf{M}^{n'}_{\mathsf{la}}$, $\mathsf{M}^n_{\mathsf{sd}}$, $\mathsf{M}^{n'}_{\mathsf{sa}}$ and $\{\mathsf{rr}\}$ are pairwise disjoint sets.

If $M \subseteq \mathsf{M}^n_{\mathsf{data}}$ and $\mathsf{m}^n_{\mathsf{data}}(i) \in M$, we write $M[i]$ for $\mathsf{m}^n_{\mathsf{data}}(i)$. If $M \subseteq \mathsf{M}^n_{\mathsf{ld}}$ and $\mathsf{m}^n_{\mathsf{ld}}(i) \in M$, we write $M[i]$ for $\mathsf{m}^n_{\mathsf{ld}}(i)$. If $M \subseteq \mathsf{M}^n_{\mathsf{la}}$ and $\mathsf{m}^n_{\mathsf{la}}(i) \in M$, we write $M[i]$ for $\mathsf{m}^n_{\mathsf{la}}(i)$. If $M \subseteq \mathsf{M}^n_{\mathsf{sd}}$ and $\mathsf{m}^n_{\mathsf{sd}}(i) \in M$, we write $M[i]$ for $\mathsf{m}^n_{\mathsf{sd}}(i)$. If $M \subseteq \mathsf{M}^n_{\mathsf{sa}}$ and $\mathsf{m}^n_{\mathsf{sa}}(i) \in M$, we write $M[i]$ for $\mathsf{m}^n_{\mathsf{sa}}(i)$. Moreover, we write $\mathbb{B}$ for the set $\{\mathsf{T}, \mathsf{F}\}$.

Let $aw, ous \geqslant 0$, $wl, nrpl, nrps > 0$ and $A_{dm} \subseteq \mathscr{A}$. Then a *strict load/store Maurer instruction set architecture* with parameters *aw*, *wl*, *ous*, *nrpl*, *nrps* and A_{dm} is a Maurer machine $H = (M, B, \mathscr{S}, \mathscr{O}, A, [\![\_]\!])$ with

$$
\begin{aligned}
M &= M_{data} \cup M_{ou} \cup M_{ld} \cup M_{la} \cup M_{sd} \cup M_{sa} \cup \{\mathsf{rr}\} \\
B &= [0, 2^{wl} - 1] \cup [0, 2^{aw} - 1] \cup \mathbb{B}
\end{aligned}
$$

$$
\begin{aligned}
\mathscr{S} &= \{S : M \to B \mid \\
&\qquad \forall m \in M_{data} \cup M_{ld} \cup M_{sd} \cdot S(m) \in [0, 2^{wl} - 1] \wedge \\
&\qquad \forall m \in M_{la} \cup M_{sa} \cdot S(m) \in [0, 2^{aw} - 1] \wedge \\
&\qquad \forall m \in M_{ou} \cdot S(m) \in \{0, 1\} \wedge S(\mathsf{rr}) \in \mathbb{B}\} \\
\mathscr{O} &= \{O_a \mid a \in A\} \\
A &= \{\mathsf{load}{:}n \mid n \in [0, nrpl - 1]\} \cup \{\mathsf{store}{:}n \mid n \in [0, nrps - 1]\} \cup A_{dm} \\
[\![a]\!] &= (O_a, \mathsf{rr}) \quad \text{for all } a \in A,
\end{aligned}
$$

where

$$
\begin{aligned}
M_{data} &= \{\mathsf{m}^{wl}_{\mathsf{data}}(i) \mid i \in [0, 2^{aw} - 1]\} \\
M_{ou} &= \{\mathsf{m}_{\mathsf{ou}}(i) \mid i \in [0, ous - 1]\} \\
M_{ld} &= \{\mathsf{m}^{wl}_{\mathsf{ld}}(i) \mid i \in [0, nrpl - 1]\} \\
M_{la} &= \{\mathsf{m}^{aw}_{\mathsf{la}}(i) \mid i \in [0, nrpl - 1]\} \\
M_{sd} &= \{\mathsf{m}^{wl}_{\mathsf{sd}}(i) \mid i \in [0, nrps - 1]\} \\
M_{sa} &= \{\mathsf{m}^{aw}_{\mathsf{sa}}(i) \mid i \in [0, nrps - 1]\}
\end{aligned}
$$

and, for all $n \in [0, nrpl - 1]$, we have $O_{\mathsf{load}:n}$ is the unique function from $\mathscr{S}$ to $\mathscr{S}$ such that for all $S \in \mathscr{S}$,

$$
\begin{aligned}
O_{\mathsf{load}:n}(S) \upharpoonright (M \setminus \{M_{ld}[n], \mathsf{rr}\}) &= S \upharpoonright (M \setminus \{M_{ld}[n], \mathsf{rr}\}) \\
O_{\mathsf{load}:n}(S)(M_{ld}[n]) &= S(M_{data}[S(M_{la}[n])]) \\
O_{\mathsf{load}:n}(S)(\mathsf{rr}) &= \mathsf{T}
\end{aligned}
$$

and, for all $n \in [0, nrps - 1]$, we have $O_{\mathsf{store}:n}$ is the unique function from $\mathscr{S}$ to $\mathscr{S}$ such that for all $S \in \mathscr{S}$,

$$
\begin{aligned}
O_{\mathsf{store}:n}(S) \upharpoonright (M \setminus \{M_{data}[S(M_{sa}[n])], \mathsf{rr}\}) &= S \upharpoonright (M \setminus \{M_{data}[S(M_{sa}[n])], \mathsf{rr}\}) \\
O_{\mathsf{store}:n}(S)(M_{data}[S(M_{sa}[n])]) &= S(M_{sd}[n]) \\
O_{\mathsf{store}:n}(S)(\mathsf{rr}) &= \mathsf{T}
\end{aligned}
$$

and, for all $a \in A_{dm}$, we have O_a is a function from $\mathscr{S}$ to $\mathscr{S}$ such that

$$IR(O_a) \subseteq M_{ou} \cup M_{ld}$$

$$OR(O_a) \subseteq M_{ou} \cup M_{la} \cup M_{sd} \cup M_{sa} \cup \{\mathsf{rr}\}.$$

We will write $\mathscr{MISA}_{\mathrm{sls}}(aw, wl, ous, nrpl, nrps, A_{dm})$ for the set of all strict load/store Maurer ISAs with parameters aw, wl, ous, $nrpl$, $nrps$ and A_{dm}.

In our opinion, load/store architectures give rise to a relatively simple interface between the data memory and the operating unit.

A strict load/store Maurer ISA is strict in the following respects:

— with data transfer between the data memory and the operating unit, a strict separation is made between data registers for loading, address registers for loading, data registers for storing, and address registers for storing;

— from these registers, only the registers of the first kind (data registers for loading) are allowed in the input regions of data manipulation operations, and only the registers of the other three kinds are allowed in the output regions of data manipulation operations;
— a data memory with a size less than the number of addresses determined by the address width is not allowed.

The first two of these conditions concern the interface between the data memory and the operating unit. We believe that they provide the most convenient interface for theoretical work relevant to the design of instruction set architectures. The third condition means we do not need to deal with addresses that do not address a memory element. Such addresses can be dealt with in many different ways, but since they all complicate the architecture considerably, we believe it is a good idea to exclude them in much theoretical work relevant to the design of instruction set architectures.

A strict separation between data registers for loading, address registers for loading, data registers for storing, and address registers for storing was also made in Cray and Thornton's design of the CDC 6600 computer (Thornton 1970), which was, arguably, the first load/store architecture to be implemented. However, in their design, data registers for loading are also allowed in the input regions of data manipulation operations.

6. Reducing the operating unit size

In a strict load/store Maurer ISA, data manipulation takes place in the operating unit. This raises questions about the consequences of changing the operating unit size. One of the questions is whether, if the operating unit size is reduced by one, it is possible with new instructions for data manipulation to transform each thread that can be applied to the original ISA into one or more threads that can each be applied to the ISA with the reduced operating unit size and that together yield the same state changes on the data memory. This question can be answered in the affirmative.

Theorem 1. Let $aw \geqslant 0$, $wl, ous, nrpl, nrps > 0$ and $A_{dm} \subseteq \mathscr{A}$. Let

$$\begin{aligned} H &= (M, B, \mathscr{S}, \mathscr{O}, A, [\![\_]\!]) \in \mathscr{MISA}_{\mathsf{sls}}(aw, wl, ous, nrpl, nrps, A_{dm}) \\ M_{data} &= \{\mathsf{m}^{wl}_{\mathsf{data}}(i) \mid i \in [0, 2^{aw} - 1]\} \\ \mathsf{bc} &= \mathsf{m}_{\mathsf{ou}}(ous - 1). \end{aligned}$$

Then there exist an $A'_{dm} \subseteq \mathscr{A}$ and an

$$H' = (M', B, \mathscr{S}', \mathscr{O}', A', [\![\_]\!]') \in \mathscr{MISA}_{\mathsf{sls}}(aw, wl, ous - 1, nrpl, nrps, A'_{dm})$$

such that for all $p \in \mathscr{T}_{\mathsf{finrec}}(A)$ there exist $p'_0, p'_1 \in \mathscr{T}_{\mathsf{finrec}}(A')$ such that

$$\begin{aligned} &\{(S \restriction M_{data}, (p \bullet_H S) \restriction M_{data}) \mid S \in \mathscr{S} \wedge S(\mathsf{bc}) = 0\} \\ &\quad = \{(S' \restriction M_{data}, (p'_0 \bullet_{H'} S') \restriction M_{data}) \mid S' \in \mathscr{S}'\} \end{aligned}$$

and

$$\begin{aligned} &\{(S \restriction M_{data}, (p \bullet_H S) \restriction M_{data}) \mid S \in \mathscr{S} \wedge S(\mathsf{bc}) = 1\} \\ &\quad = \{(S' \restriction M_{data}, (p'_1 \bullet_{H'} S') \restriction M_{data}) \mid S' \in \mathscr{S}'\}. \end{aligned}$$

Notice that bc is the operating unit memory element of H that is missing in H'. In the proof of Theorem 1 given below, we take A'_{dm} such that, for each instruction a in A_{dm}, there are four instructions $a(0)$, $a(1)$, $\overline{a}(0)$ and $\overline{a}(1)$ in A'_{dm}. The operations $O_{a(0)}$ and $O_{a(1)}$ affect the memory elements of H' in the same way as O_a would affect them if the content of the missing operating unit memory element was 0 or 1, respectively. The effect that O_a would have on the missing operating unit memory element is made available by $O_{\overline{a}(0)}$ and $O_{\overline{a}(1)}$, respectively. They do nothing but reply F if the content of the missing operating unit memory element would become 0 and T if the content of the missing operating unit memory element would become 1.

Proof of Theorem 1. Instead of proving the statement directly, we prove that there exist an $A'_{dm} \subseteq \mathscr{A}$ and an

$$H' = (M', B, \mathscr{S}', \mathscr{O}', A', [\![\_]\!]') \in \mathscr{MISA}_{\mathrm{sls}}(aw, wl, ous - 1, nrpl, nrps, A'_{dm})$$

such that for all $p \in \mathscr{T}_{\mathsf{finrec}}(A)$ there exist $p'_0, p'_1 \in \mathscr{T}_{\mathsf{finrec}}(A')$ such that

$$\begin{aligned}&\{(S \restriction (M' \setminus \{\mathsf{rr}\}), (p \bullet_H S) \restriction (M' \setminus \{\mathsf{rr}\})) \mid S \in \mathscr{S} \wedge S(\mathsf{bc}) = 0\}\\ &\quad = \{(S' \restriction (M' \setminus \{\mathsf{rr}\}), (p'_0 \bullet_{H'} S') \restriction (M' \setminus \{\mathsf{rr}\})) \mid S' \in \mathscr{S}'\}\end{aligned}$$

and

$$\begin{aligned}&\{(S \restriction (M' \setminus \{\mathsf{rr}\}), (p \bullet_H S) \restriction (M' \setminus \{\mathsf{rr}\})) \mid S \in \mathscr{S} \wedge S(\mathsf{bc}) = 1\}\\ &\quad = \{(S' \restriction (M' \setminus \{\mathsf{rr}\}), (p'_1 \bullet_{H'} S') \restriction (M' \setminus \{\mathsf{rr}\})) \mid S' \in \mathscr{S}'\}.\end{aligned}$$

This is sufficient because $M_{data} \subseteq M' \setminus \{\mathsf{rr}\}$.

We take

$$A'_{dm} = \{a(k), \overline{a}(k) \mid a \in A_{dm} \wedge k \in \{0, 1\}\},$$

and

$$H' = (M', B, \mathscr{S}', \mathscr{O}', A', [\![\_]\!]')$$

such that for each $a \in A_{dm}$ and $k \in \{0, 1\}$, we have $O_{a(k)}$ and $O_{\overline{a}(k)}$ are the unique functions from $\mathscr{S}'$ to $\mathscr{S}'$ such that for all $S' \in \mathscr{S}'$:

$$O_{a(k)}(S') \;=\; O_a(\rho_k(S')) \restriction M'$$

and

$$\begin{aligned}O_{\overline{a}(k)}(S') \restriction (M' \setminus \{\mathsf{rr}\}) \;&=\; S' \restriction (M' \setminus \{\mathsf{rr}\})\\ O_{\overline{a}(k)}(S')(\mathsf{rr}) \;&=\; \gamma(O_a(\rho_k(S'))(\mathsf{bc}))\end{aligned}$$

where, for each $k \in \{0, 1\}$, we have ρ_k is the unique function from $\mathscr{S}'$ to $\mathscr{S}$ such that

$$\begin{aligned}\rho_k(S') \restriction M' &= S'\\ \rho_k(S')(\mathsf{bc}) \;&= k\end{aligned}$$

and $\gamma : \{0, 1\} \to \mathbb{B}$ is defined by

$$\begin{aligned}\gamma(0) &= \mathsf{F}\\ \gamma(1) &= \mathsf{T}.\end{aligned}$$

We restrict ourselves to $p \in \{\langle X|E\rangle \mid E \in \mathscr{E}^{\mathsf{lin}}_{\mathsf{fin}}(A) \wedge X \in V(E)\}$ because each term from $\mathscr{T}_{\mathsf{finrec}}(A)$ can be proved to be equal to some constant from this set by means of the axioms of BTA+REC.

We define transformation functions

$$\phi_k : \{\langle X|E\rangle \mid E \in \mathscr{E}^{\mathsf{lin}}_{\mathsf{fin}}(A) \wedge X \in V(E)\} \to \{\langle X|E\rangle \mid E \in \mathscr{E}^{\mathsf{lin}}_{\mathsf{fin}}(A') \wedge X \in V(E)\},$$

for $k \in \{0,1\}$, as follows:

$$\phi_k(\langle X|E\rangle) \;=\; \langle X_k|\phi'_k(E)\rangle,$$

where $\phi'_k : \mathscr{E}^{\mathsf{lin}}_{\mathsf{fin}}(A) \to \mathscr{E}^{\mathsf{lin}}_{\mathsf{fin}}(A')$, for $k \in \{0,1\}$ is defined as follows:

$$\begin{aligned}
\phi'_k(\{X = \mathsf{S}\}) &= \{X_k = \mathsf{S}\} \\
\phi'_k(\{X = \mathsf{D}\}) &= \{X_k = \mathsf{D}\} \\
\phi'_k(\{X = Y \trianglelefteq a \trianglerighteq Z\}) &= \{X_k = Y_k \trianglelefteq a \trianglerighteq Z_k\} && \text{if } a \notin A_{dm} \\
\phi'_k(\{X = Y \trianglelefteq a \trianglerighteq Z\}) &= \{X_k = X'_k \trianglelefteq \overline{a}(k) \trianglerighteq X''_k, \\
&\qquad X'_k = Y_1 \trianglelefteq a(k) \trianglerighteq Z_1, \\
&\qquad X''_k = Y_0 \trianglelefteq a(k) \trianglerighteq Z_0\} && \text{if } a \in A_{dm} \\
\phi'_k(E' \cup E'') &= \phi'_k(E') \cup \phi'_k(E'').
\end{aligned}$$

Here, for each variable X, the new variables X_0, X'_0, X''_0, X_1, X'_1 and X''_1 are taken such that:

(i) they are pairwise different variables;
(ii) for each variable Y different from X, we have $\{X_0, X'_0, X''_0, X_1, X'_1, X''_1\}$ and $\{Y_0, Y'_0, Y''_0, Y_1, Y'_1, Y''_1\}$ are disjoint sets.

Let $p \in \{\langle X|E\rangle \mid E \in \mathscr{E}^{\mathsf{lin}}_{\mathsf{fin}}(A) \wedge X \in V(E)\}$. Let $S \in \mathscr{S}$ and $S' \in \mathscr{S}'$ be such that $S \upharpoonright M' = S'$, let (p_i, S_i) be the $(i+1)$th element in the full path of (p, S) on H, and let (p'_i, S'_i) be the $(i+1)$th element in the full path of $(\phi_{S(\mathsf{bc})}(p), S')$ on H' whose first component does not equal $p \trianglelefteq a(k) \trianglerighteq q$ for any $p, q \in \mathscr{T}_{\mathsf{finrec}}$, $a \in A_{dm}$ and $k \in \{0,1\}$. Moreover, let a_i be the unique $a \in A$ such that $p_i = p \trianglelefteq a \trianglerighteq q$ for some $p, q \in \mathscr{T}_{\mathsf{finrec}}$. It is then easy to prove by induction on i that if $a_i \in A_{dm}$, then

$$O_{a_i}(S_i)(\mathsf{bc}) \;=\; \gamma^{-1}(O_{\overline{a}_i(S_i(\mathsf{bc}))}(S'_i)(\mathsf{rr})) \tag{1}$$

$$O_{a_i}(S_i)(\mathsf{rr}) \;=\; O_{a_i(S_i(\mathsf{bc}))}(O_{\overline{a}_i(S_i(\mathsf{bc}))}(S'_i))(\mathsf{rr}) \tag{2}$$

(if $i + 1 < \|(p, S)\|_H$ when p converges from S on H). Now, using (1) and (2), it is easy to prove by induction on i that:

$$\begin{aligned}
\phi_{S_i(\mathsf{bc})}(p_i) &= p'_i \\
S_i \upharpoonright (M \setminus \{\mathsf{rr}\}) &= \rho_{S_i(\mathsf{bc})}(S'_i) \upharpoonright (M \setminus \{\mathsf{rr}\})
\end{aligned}$$

(if $i < \|(p, S)\|_H$ when p converges from S on H). The result then follows immediately. □

The proof of Theorem 1 gives us some upper bounds:

— for each thread that can be applied to the original ISA, the number of threads that can together produce the same state changes on the data memory of the ISA with the reduced operating unit does not have to be more than 2;
— the number of states of the new threads does not have to be more than 6 times the number of states of the original thread;
— the number of steps that the new threads take to produce some state change does not have to be more than 2 times the number of steps that the original thread takes to produce that state change;
— the number of instructions of the ISA with the reduced operating unit does not have to be more than 4 times the number of instructions of the original ISA.

Moreover, the proof indicates that more efficient new threads are possible: equations $X = Y \trianglelefteq a \trianglerighteq Z$ with $a \in A_{dm}$ can be treated as if $a \notin A_{dm}$ when the missing operating unit memory element is not in $IR(O_a)$.

As a corollary of the proof of Theorem 1, we have that only one transformed thread is needed if the input region of the operation associated with the first instruction performed by the original thread does not include the operating unit memory element that is missing in the ISA with the reduced operating unit size.

Corollary 1. Let $aw \geqslant 0$, $wl, ous, nrpl, nrps > 0$ and $A_{dm} \subseteq \mathscr{A}$. Let:

$$\begin{aligned} H &= (M, B, \mathscr{S}, \mathscr{O}, A, [\![\_]\!]) \in \mathscr{MISA}_{\mathrm{sls}}(aw, wl, ous, nrpl, nrps, A_{dm}) \\ M_{data} &= \{\mathsf{m}^{wl}_{\mathsf{data}}(i) \mid i \in [0, 2^{aw} - 1]\} \\ \mathsf{bc} &= \mathsf{m}_{\mathsf{ou}}(ous - 1). \end{aligned}$$

Moreover, let

$$\mathscr{T}' = \{q \trianglelefteq a \trianglerighteq r \mid q, r \in \mathscr{T}_{\mathsf{finrec}}(A) \wedge a \in A \wedge \mathsf{bc} \in IR(O_a)\}.$$

Then there exist an $A'_{dm} \subseteq \mathscr{A}$ and an

$$H' = (M', B, \mathscr{S}', \mathscr{O}', A', [\![\_]\!]') \in \mathscr{MISA}_{\mathrm{sls}}(aw, wl, ous - 1, nrpl, nrps, A'_{dm})$$

such that for all $p \in \mathscr{T}_{\mathsf{finrec}}(A) \setminus \mathscr{T}'$ there exists a $p' \in \mathscr{T}_{\mathsf{finrec}}(A')$ such that

$$\begin{aligned} &\{(S \restriction M_{data}, (p \bullet_H S) \restriction M_{data}) \mid S \in \mathscr{S}\} \\ &\quad = \{(S' \restriction M_{data}, (p' \bullet_{H'} S') \restriction M_{data}) \mid S' \in \mathscr{S}'\}. \end{aligned}$$

As another corollary of the proof of Theorem 1, if the operating unit size is reduced to zero, it is still possible to transform each thread that can be applied to the original ISA into a number of threads that can each be applied to the ISA with the reduced operating unit size and that together yield the same state changes on the data memory.

Corollary 2. Let $aw \geqslant 0$, $wl, ous, nrpl, nrps > 0$ and $A_{dm} \subseteq \mathscr{A}$. Let

$$\begin{aligned} H &= (M, B, \mathscr{S}, \mathscr{O}, A, [\![\_]\!]) \in \mathscr{MISA}_{\mathrm{sls}}(aw, wl, ous, nrpl, nrps, A_{dm}) \\ M_{data} &= \{\mathsf{m}^{wl}_{\mathsf{data}}(i) \mid i \in [0, 2^{aw} - 1]\} \\ M_{ou} &= \{\mathsf{m}_{\mathsf{ou}}(i) \mid i \in [0, ous - 1]\}. \end{aligned}$$

Then there exist an $A'_{dm} \subseteq \mathscr{A}$ and an

$$H' = (M', B, \mathscr{S}', \mathscr{O}', A', [\![\_]\!]') \in \mathscr{MISA}_{\mathrm{sls}}(aw, wl, 0, nrpl, nrps, A'_{dm})$$

such that for all $p \in \mathscr{T}_{\mathsf{finrec}}(A)$ and $S_{ou} \in \{S \restriction M_{ou} \mid S \in \mathscr{S}\}$ there exists a $p' \in \mathscr{T}_{\mathsf{finrec}}(A')$ such that

$$\begin{aligned}&\{(S \restriction M_{data}, (p \bullet_H S) \restriction M_{data}) \mid S \in \mathscr{S} \wedge S \restriction M_{ou} = S_{ou}\}\\ &\quad = \{(S' \restriction M_{data}, (p' \bullet_{H'} S') \restriction M_{data}) \mid S' \in \mathscr{S}'\}.\end{aligned}$$

Let M_{ou} be as in Corollary 2. Then the cardinality of $\{S \restriction M_{ou} \mid S \in \mathscr{S}\}$ is 2^{ous}. Therefore, if the operating unit size is reduced to zero, at most 2^{ous} transformed threads are needed for each thread that can be applied to the original ISA. Notice that Corollary 2 does not go through if the number of states of the new threads is bounded.

7. Thread powered function classes

A simple calculation shows that for a strict load/store Maurer ISA with address width aw and word length wl, the number of possible transformations on the states of the data memory is $2^{(2^{(2^{aw} \cdot wl + aw)} \cdot wl)}$. This raises questions concerning the possibility of achieving all these state transformation by applying a thread to a strict load/store Maurer ISA with this address width and word length. One of the questions is how this possibility depends on the operating unit size of the ISAs, the size of the instruction set of the ISAs and the maximal number of states of the threads. This leads us to introduce the concept of a thread powered function class.

The concept of a thread powered function class is parametrised by:

— an address width aw;
— a word length wl;
— an operating unit size ous;
— an instruction set size iss;
— a state space bound ssb;
— a working area flag waf;

where $aw, ous \geqslant 0$, $wl, iss, ssb > 0$ and $waf \in \mathbb{B}$.

The instruction set size iss is the number of basic instructions, excluding load and store instructions. To simplify the setting, we will only consider the case where there is one load instruction and one store instruction. The state space bound ssb is a bound on the number of states of the thread that is applied. The working area flag waf indicates whether a part of the data memory is taken as a working area. A part of the data memory is taken as a working area if we are not interested in the state transformations with respect to that part. To simplify the setting, we always set aside half of the data memory for working area if a working area is in order.

Intuitively, the thread powered function class with parameters aw, wl, ous, iss, ssb and waf are the transformations on the states of the data memory, or the first half of the data memory, depending on waf, that can be achieved by applying threads with no more than ssb states to a strict load/store Maurer ISA of which the address width is aw, the word length is wl, the operating unit size is ous, the number of register pairs for load

instructions is 1, the number of register pairs for store instructions is 1, and the cardinality of the set of instructions for data manipulation is *iss*. From now on, we will use the term *external memory* for the data memory if $waf = \mathsf{F}$ and for the first half of the data memory if $waf = \mathsf{T}$. Moreover, if $waf = \mathsf{T}$, we will use the term *internal memory* for the second half of the data memory.

For $aw \geqslant 0$ and $wl > 0$, we define $\mathsf{M}_{\mathsf{data}}^{aw,wl}$, $\mathsf{S}_{\mathsf{data}}^{aw,wl}$ and $\mathsf{T}_{\mathsf{data}}^{aw,wl}$ as follows:

$$\begin{aligned}\mathsf{M}_{\mathsf{data}}^{aw,wl} &= \{\mathsf{m}_{\mathsf{data}}^{wl}(i) \mid i \in [0, 2^{aw} - 1]\}\\ \mathsf{S}_{\mathsf{data}}^{aw,wl} &= \{S \mid S : \mathsf{M}_{\mathsf{data}}^{aw,wl} \to [0, 2^{wl} - 1]\}\\ \mathsf{T}_{\mathsf{data}}^{aw,wl} &= \{T \mid T : \mathsf{S}_{\mathsf{data}}^{aw,wl} \to \mathsf{S}_{\mathsf{data}}^{aw,wl}\}.\end{aligned}$$

Here:

— $\mathsf{M}_{\mathsf{data}}^{aw,wl}$ is the data memory of a strict load/store Maurer ISA with address width *aw* and word length *wl*;
— $\mathsf{S}_{\mathsf{data}}^{aw,wl}$ is the set of possible states of that data memory;
— $\mathsf{T}_{\mathsf{data}}^{aw,wl}$ is the set of possible transformations on those states.

Let $aw, ous \geqslant 0$ and $wl, iss, ssb > 0$. Let $waf \in \mathbb{B}$ be such that $waf = \mathsf{F}$ if $aw = 0$. Then the *thread powered function class* with parameters *aw*, *wl*, *ous*, *iss*, *ssb* and *waf*, written $\mathscr{TPFC}(aw, wl, ous, iss, ssb, waf)$, is the subset of $\mathsf{T}_{\mathsf{data}}^{aw,wl}$ defined by

$$\begin{aligned}T \in \mathscr{TPFC}(aw, wl, ous, iss, ssb, waf)\\ \Leftrightarrow \exists A_{dm} \subseteq \mathscr{A} \cdot\\ \exists H \in \mathscr{MIS}\mathscr{A}_{\mathrm{sls}}(aw, wl, ous, 1, 1, A_{dm}) \cdot\\ \exists p \in \mathscr{T}_{\mathsf{finrec}}(A_H) \cdot\\ \big(card(A_{dm}) = iss \wedge card(Res(p)) \leqslant ssb \wedge\\ \forall S \in \mathscr{S}_H \cdot\\ \big(\big(waf = \mathsf{F} \Rightarrow T(S \restriction \mathsf{M}_{\mathsf{data}}^{aw,wl}) = (p \bullet_H S) \restriction \mathsf{M}_{\mathsf{data}}^{aw,wl}\big) \wedge\\ \big(waf = \mathsf{T} \Rightarrow\\ T(S \restriction \mathsf{M}_{\mathsf{data}}^{aw,wl}) \restriction \mathsf{M}_{\mathsf{data}}^{aw-1,wl} = (p \bullet_H S) \restriction \mathsf{M}_{\mathsf{data}}^{aw-1,wl}\big)\big)\big).\end{aligned}$$

We say that $\mathscr{TPFC}(aw, wl, ous, iss, ssb, waf)$ is *complete* if

$$\mathscr{TPFC}(aw, wl, ous, iss, ssb, waf) = \mathsf{T}_{\mathsf{data}}^{aw,wl}.$$

The following theorem states that $\mathscr{TPFC}(aw, wl, ous, iss, ssb, waf)$ is complete if $ous = 2^{aw} \cdot wl + aw + 1$, $iss = 5$ and $ssb = 8$. Because $2^{aw} \cdot wl$ is the data memory size, that is, the number of bits that the data memory contains, this means that completeness can be obtained using 5 data manipulation instructions and threads whose number of states is less than or equal to 8 by taking the operating unit size to be slightly greater than the data memory size.

Theorem 2. Let $aw \geqslant 0$, $wl > 0$ and $waf \in \mathbb{B}$, and let $dms = 2^{aw} \cdot wl$. Then $\mathscr{TPFC}(aw, wl, dms + aw + 1, 5, 8, waf)$ is complete.

The idea behind the proof of Theorem 2 is:

1 The content of the whole data memory is copied data memory element by data memory element via the load data register to the operating unit.

2 The intended state transformation is then applied to the copy in the operating unit.
3 Finally, the result is copied back data memory element by data memory element via the store data register to the data memory.

The data manipulation instructions used to accomplish this are an initialisation instruction, a pre-load instruction, a post-load instruction, a pre-store instruction and a transformation instruction:

— The pre-load instruction is used to update the load address register before a data memory element is loaded.
— The post-load instruction is used to store the content of the load data register to the operating unit after a data memory element has been loaded.
— The pre-store instruction is used to update the store address register and to load the content of the store data register from the operating unit before a data memory element is stored.
— The transformation instruction is used to apply the intended state transformation to the copy in the operating unit.

Proof of Theorem 2. For convenience, we define

$$\begin{aligned}
M_{data} &= \{\mathsf{m}^{wl}_{\mathsf{data}}(i) \mid i \in [0, 2^{aw} - 1]\} \\
M_{ou} &= \{\mathsf{m}_{\mathsf{ou}}(j) \mid j \in [0, dms + aw]\} \\
M^d_{ou} &= \{\mathsf{m}_{\mathsf{ou}}(j) \mid j \in [0, dms - 1]\} \\
M^a_{ou} &= \{\mathsf{m}_{\mathsf{ou}}(j) \mid j \in [dms, dms + aw]\} \\
M^d_{ou}\langle i\rangle &= \{\mathsf{m}_{\mathsf{ou}}(j) \mid j \in [i \cdot wl, (i+1) \cdot wl - 1]\} \text{ for } i \in [0, 2^{aw} - 1] \\
\mathsf{ldr} &= \mathsf{m}^{wl}_{\mathsf{ld}}(0) \\
\mathsf{sdr} &= \mathsf{m}^{wl}_{\mathsf{sd}}(0) \\
\mathsf{lar} &= \mathsf{m}^{aw}_{\mathsf{la}}(0) \\
\mathsf{sar} &= \mathsf{m}^{aw}_{\mathsf{sa}}(0).
\end{aligned}$$

We have $M^d_{ou} = \bigcup_{i \in [0,2^{aw}-1]} M^d_{ou}\langle i\rangle$ and $M_{ou} = M^d_{ou} \cup M^a_{ou}$.

We have to deal with the binary representations of natural numbers in the operating unit. The set of possible binary representations of natural numbers in the operating unit is

$$\mathscr{R} = \bigcup_{n \in [0, dms+aw], m \in [n, dms+aw]} \{R \mid R : \{\mathsf{m}_{\mathsf{ou}}(i) \mid i \in [n, m]\} \to \{0, 1\}\}.$$

For each $R \in \mathscr{R}$, the natural number for which R is a binary representation is given by the function $v : \mathscr{R} \to \mathbb{N}$ defined by

$$v(R) = \textstyle\sum_{i \text{ s.t. } \mathsf{m}_{\mathsf{ou}}(i) \in \mathrm{dom}(R)} R(\mathsf{m}_{\mathsf{ou}}(i)) \cdot 2^{i - \min\{j \mid \mathsf{m}_{\mathsf{ou}}(j) \in \mathrm{dom}(R)\}}.$$

To prove the theorem, we take a fixed but arbitrary $T \in \mathsf{T}^{aw,wl}_{\mathsf{data}}$ and show that $T \in \mathscr{TPFC}(aw, wl, dms + aw + 1, 5, 8, waf)$.

We take

$$A_{dm} = \{\mathsf{init}, \mathsf{preload}, \mathsf{postload}, \mathsf{prestore}, \mathsf{transform}\}$$

and

$$H = (M, B, \mathscr{S}, \mathscr{O}, A, [\![\_]\!]) \in \mathscr{MISA}_{\mathrm{sls}}(aw, wl, dms + aw + 1, 1, 1, A_{dm})$$

such that O_{init}, O_{preload}, O_{postload}, O_{prestore}, and $O_{\mathsf{transform}}$ are the unique functions from $\mathscr{S}$ to $\mathscr{S}$ such that for all $S \in \mathscr{S}$:

$$\begin{aligned}
O_{\mathsf{init}}(S) \upharpoonright (M \setminus (M^a_{ou} \cup \{\mathsf{rr}\})) &= S \upharpoonright (M \setminus (M^a_{ou} \cup \{\mathsf{rr}\})) \\
v(O_{\mathsf{init}}(S) \upharpoonright M^a_{ou}) &= 0 \\
O_{\mathsf{init}}(S)(\mathsf{rr}) &= \mathsf{T}
\end{aligned}$$

$$\begin{aligned}
O_{\mathsf{preload}}(S) \upharpoonright (M \setminus (M^a_{ou} \cup \{\mathsf{lar}\} \cup \{\mathsf{rr}\})) &= S \upharpoonright (M \setminus (M^a_{ou} \cup \{\mathsf{lar}\} \cup \{\mathsf{rr}\})) \\
v(O_{\mathsf{preload}}(S) \upharpoonright M^a_{ou}) &= v(S \upharpoonright M^a_{ou}) + 1 && \text{if } v(S \upharpoonright M^a_{ou}) < 2^{aw} \\
O_{\mathsf{preload}}(S) \upharpoonright M^a_{ou} &= S \upharpoonright M^a_{ou} && \text{if } v(S \upharpoonright M^a_{ou}) \geqslant 2^{aw} \\
O_{\mathsf{preload}}(S)(\mathsf{lar}) &= v(S \upharpoonright M^a_{ou}) && \text{if } v(S \upharpoonright M^a_{ou}) < 2^{aw} \\
O_{\mathsf{preload}}(S)(\mathsf{lar}) &= S(\mathsf{lar}) && \text{if } v(S \upharpoonright M^a_{ou}) \geqslant 2^{aw} \\
O_{\mathsf{preload}}(S)(\mathsf{rr}) &= \mathsf{T} && \text{if } v(S \upharpoonright M^a_{ou}) < 2^{aw} \\
O_{\mathsf{preload}}(S)(\mathsf{rr}) &= \mathsf{F} && \text{if } v(S \upharpoonright M^a_{ou}) \geqslant 2^{aw}
\end{aligned}$$

$$\begin{aligned}
O_{\mathsf{postload}}(S) \upharpoonright (M \setminus (M^d_{ou}\langle v(S \upharpoonright M^a_{ou})\rangle \cup \{\mathsf{rr}\})) &= S \upharpoonright (M \setminus (M^d_{ou}\langle v(S \upharpoonright M^a_{ou})\rangle \cup \{\mathsf{rr}\})) \\
v(O_{\mathsf{postload}}(S) \upharpoonright M^d_{ou}\langle v(S \upharpoonright M^a_{ou})\rangle) &= S(\mathsf{ldr}) \\
O_{\mathsf{postload}}(S)(\mathsf{rr}) &= \mathsf{T}
\end{aligned}$$

$$\begin{aligned}
O_{\mathsf{prestore}}(S) \upharpoonright (M \setminus (M^a_{ou} \cup \{\mathsf{sar}, \mathsf{sdr}\} \cup \{\mathsf{rr}\})) &= S \upharpoonright (M \setminus (M^a_{ou} \cup \{\mathsf{sar}, \mathsf{sdr}\} \cup \{\mathsf{rr}\})) \\
v(O_{\mathsf{prestore}}(S) \upharpoonright M^a_{ou}) &= v(S \upharpoonright M^a_{ou}) + 1 && \text{if } v(S \upharpoonright M^a_{ou}) < 2^{aw} \\
O_{\mathsf{prestore}}(S) \upharpoonright M^a_{ou} &= S \upharpoonright M^a_{ou} && \text{if } v(S \upharpoonright M^a_{ou}) \geqslant 2^{aw} \\
O_{\mathsf{prestore}}(S)(\mathsf{sar}) &= v(S \upharpoonright M^a_{ou}) && \text{if } v(S \upharpoonright M^a_{ou}) < 2^{aw} \\
O_{\mathsf{prestore}}(S)(\mathsf{sar}) &= S(\mathsf{sar}) && \text{if } v(S \upharpoonright M^a_{ou}) \geqslant 2^{aw} \\
O_{\mathsf{prestore}}(S)(\mathsf{sdr}) &= v(S \upharpoonright M^d_{ou}\langle v(S \upharpoonright M^a_{ou})\rangle) \\
& && \text{if } v(S \upharpoonright M^a_{ou}) < 2^{aw} \\
O_{\mathsf{prestore}}(S)(\mathsf{sdr}) &= S(\mathsf{sdr}) && \text{if } v(S \upharpoonright M^a_{ou}) \geqslant 2^{aw} \\
O_{\mathsf{prestore}}(S)(\mathsf{rr}) &= \mathsf{T} && \text{if } v(S \upharpoonright M^a_{ou}) < 2^{aw} \\
O_{\mathsf{prestore}}(S)(\mathsf{rr}) &= \mathsf{F} && \text{if } v(S \upharpoonright M^a_{ou}) \geqslant 2^{aw}
\end{aligned}$$

$$\begin{aligned}
O_{\mathsf{transform}}(S) \upharpoonright (M \setminus (M_{ou} \cup \{\mathsf{rr}\})) &= S \upharpoonright (M \setminus (M_{ou} \cup \{\mathsf{rr}\})) \\
O_{\mathsf{transform}}(S) \upharpoonright M^d_{ou} &= T'(S \upharpoonright M^d_{ou}) \\
v(O_{\mathsf{transform}}(S) \upharpoonright M^a_{ou}) &= 0 \\
O_{\mathsf{transform}}(S)(\mathsf{rr}) &= \mathsf{T}
\end{aligned}$$

where T' is the unique function from $\{S \upharpoonright M^d_{ou} \mid S \in \mathscr{S}\}$ to $\{S \upharpoonright M^d_{ou} \mid S \in \mathscr{S}\}$ such that for all $S^d_{ou} \in \{S \upharpoonright M^d_{ou} \mid S \in \mathscr{S}\}$, there exists an $S_{data} \in \{S \upharpoonright M_{data} \mid S \in \mathscr{S}\}$ such that

$$\begin{aligned}
&\forall i \in [0, 2^{aw} - 1] \cdot \\
&\quad (v(S^d_{ou} \upharpoonright M^d_{ou}\langle i\rangle) = S_{data}(M_{data}[i]) \wedge \\
&\quad\ v(T'(S^d_{ou}) \upharpoonright M^d_{ou}\langle i\rangle) = T(S_{data})(M_{data}[i])).
\end{aligned}$$

Moreover, we take $p \in \mathscr{T}_{\mathsf{finrec}}(A)$ such that $p = \langle X|E\rangle$ with E consisting of the following equations:

$$\begin{aligned} X &= \mathsf{init} \circ Y \\ Y &= (\mathsf{load}{:}0 \circ \mathsf{postload} \circ Y) \trianglelefteq \mathsf{preload} \trianglerighteq (\mathsf{transform} \circ Z) \\ Z &= (\mathsf{store}{:}0 \circ Z) \trianglelefteq \mathsf{prestore} \trianglerighteq \mathsf{S}. \end{aligned}$$

Let $S \in \mathscr{S}$ and (p_i, S_i) be the $(i+1)$th element in the full path of $(\langle X|E\rangle, S)$ on H whose first component equals $\langle X|E\rangle$, $\langle Y|E\rangle$, $\langle Z|E\rangle$ or S. It is then easy to prove by induction on i that

$$i = 1 \Rightarrow p_i = \langle Y|E\rangle \wedge v(S_i \upharpoonright M^a_{ou}) = 0$$

$$\begin{aligned} &i \in [2, 2^{aw}+2] \Rightarrow \\ &\quad p_i = \langle Y|E\rangle \wedge \\ &\quad \forall j \in [0, i-2] \cdot v(S_i \upharpoonright M^d_{ou}\langle j\rangle) = S(M_{data}[j]) \wedge \\ &\quad v(S_i \upharpoonright M^a_{ou}) = i-1 \end{aligned}$$

$$\begin{aligned} &i = 2^{aw}+3 \Rightarrow \\ &\quad p_i = \langle Z|E\rangle \wedge \\ &\quad \forall j \in [0, 2^{aw}-1] \cdot v(S_i \upharpoonright M^d_{ou}\langle j\rangle) = T(S \upharpoonright M_{data})(M_{data}[j]) \wedge \\ &\quad v(S_i \upharpoonright M^a_{ou}) = 0 \end{aligned}$$

$$\begin{aligned} &i \in [2^{aw}+4, 2^{aw+1}+4] \Rightarrow \\ &\quad p_i = \langle Z|E\rangle \wedge \\ &\quad \forall j \in [0, i-(2^{aw}+4)] \cdot S_i(M_{data}[j]) = T(S \upharpoonright M_{data})(M_{data}[j]) \wedge \\ &\quad v(S_i \upharpoonright M^a_{ou}) = i-(2^{aw}+3) \end{aligned}$$

$$i = 2^{aw+1}+5 \Rightarrow p_i = \mathsf{S} \wedge S_i \upharpoonright M_{data} = T(S \upharpoonright M_{data}).$$

Hence, $(\langle X|E\rangle \bullet_H S) \upharpoonright M_{data} = T(S \upharpoonright M_{data})$. That is, T can be achieved by applying $\langle X|E\rangle$ to H. □

As a corollary of the proof of Theorem 2, when $waf = \mathsf{T}$, completeness can also be obtained if we take about half the external memory size as the operating unit size.

Corollary 3. Let $aw > 0$ and $wl > 0$, and let $ems = 2^{aw-1} \cdot wl$. Then $\mathscr{TPFC}(aw, wl, ems + aw, 5, 8, \mathsf{T})$ is complete.

As a corollary of the proofs of Theorems 1 and 2, we have that completeness can even be obtained if we take the operating unit size to be zero. However, this may require quite a large number of data manipulation instructions and threads with quite a large number of states.

Corollary 4. Let $aw \geqslant 0$ and $wl > 0$, let $waf \in \mathbb{B}$ be such that $waf = \mathsf{F}$ if $aw = 0$, and let $dms = 2^{aw} \cdot wl$. Then $\mathscr{TPFC}(aw, wl, 0, 5 \cdot 4^{dms+aw+1}, 8 \cdot 6^{dms+aw+1}, waf)$ is complete.

8. On incomplete thread powered function classes

From Corollary 4, we know that it is possible to achieve all transformations on the states of the external memory of a strict load/store Maurer ISA with given address width and word length even if the operating unit size is zero. However, this may require quite a large number of data manipulation instructions and threads with quite a large number of states. This leads us to ask whether the operating unit size of the ISAs, the size of the instructions set of the ISAs and the maximal number of states of the threads can be taken such that it is impossible to achieve all transformations on the states of the external memory.

Theorem 3 will address this question, but first we give a lemma that will be used in its proof.

Lemma 1. Let $aw > 1$ and $wl, ous, iss, ssb > 0$, and let $ems = 2^{aw-1} \cdot wl$. Then $\mathscr{TPFC}(aw, wl, ous, iss, ssb, \mathsf{T})$ is not complete if $ous \leqslant ems/2$ and $iss \leqslant 2^{ems/2}$ and there are no more than 2^{ems} threads that can be applied to the members of $\bigcup_{A_{dm} \subseteq \mathscr{A}} \mathscr{MISA}_{\mathrm{sls}}(aw, wl, ous, 1, 1, A_{dm})$.

Proof. If the operating unit size is no greater than $ems/2$, then no more than

$$(2^{ems/2})^{(2^{ems/2})}$$

transformations on the states of the operating unit can be associated with one data manipulation instruction. Because the load and store instructions are interpreted in a fixed way, it follows that if there are no more than $2^{ems/2}$ data manipulation instructions, then no more than

$$\left((2^{ems/2})^{(2^{ems/2})}\right)^{(2^{ems/2})}$$

transformations on the states of the external memory can be achieved with one thread. Hence, if no more than 2^{ems} threads can be applied, no more than

$$\left((2^{ems/2})^{(2^{ems/2})}\right)^{(2^{ems/2})} \cdot 2^{ems}$$

transformations on the states of the external memory can be achieved. Using elementary arithmetic, we easily establish that

$$\left((2^{ems/2})^{(2^{ems/2})}\right)^{(2^{ems/2})} \cdot 2^{ems} < (2^{ems})^{(2^{ems})}.$$

It follows that $\mathscr{TPFC}(aw, wl, ous, iss, ssb, \mathsf{T})$ is not complete because the number of transformations that are possible on the states of the external memory is $(2^{ems})^{(2^{ems})}$. □

In Lemma 1, the bound on the number of threads that can be applied seems to appear out of the blue. However, it is the number of threads that can at most be represented in the internal memory: even with the most efficient representations, we cannot have more than one thread per state of the internal memory.

The following theorem states that $\mathscr{TPFC}(aw, wl, ous, iss, ssb, \mathsf{T})$ is not complete if the operating unit size is no greater than half the external memory size, the instruction set size is no greater than $2^{wl} - 4$ and the maximal number of states of the threads is no

greater than 2^{aw-2}. Notice that 2^{wl} is the number of instructions that can be represented in memory elements with word length wl and that 2^{aw-2} is half the number of memory elements in the internal memory.

Theorem 3. Let $aw, wl > 1$ and $ous, iss, ssb > 0$, and let $ems = 2^{aw-1} \cdot wl$. Then $\mathcal{TPFC}(aw, wl, ous, iss, ssb, \mathsf{T})$ is not complete if $ous \leqslant ems/2$ and $iss \leqslant 2^{wl} - 4$ and $ssb \leqslant 2^{aw-2}$.

Proof. The number of threads with at most ssb states does not exceed

$$\big((iss + 2) \cdot ssb^2 + 2\big)^{ssb}$$

(recall that there is also one load instruction and one store instruction). Because $ssb > 0$,

$$\big((iss + 2) \cdot ssb^2 + 2\big)^{ssb} \leqslant \big((iss + 4) \cdot ssb^2\big)^{ssb}.$$

Using elementary arithmetic, we can easily establish that

$$\big((iss + 4) \cdot ssb^2\big)^{ssb} \leqslant 2^{ems}.$$

Consequently, there are no more than 2^{ems} threads with at most ssb states. From this, and the fact that $ous \leqslant ems/2$ and $iss < 2^{ems/2}$, it follows from Lemma 1 that $\mathcal{TPFC}(aw, wl, ous, iss, ssb, \mathsf{T})$ is not complete. □

9. Conclusions

In Bergstra and Middelburg (2007; 2008), we worked towards a formal approach to micro-architecture design based on Maurer machines and basic thread algebra. In those papers, we made hardly any assumptions about the instruction set architectures for which new micro-architectures are designed, but we put forward strict load/store Maurer instruction set architectures as the preferred instruction set architectures. In the current paper, we have established general properties of strict load/store Maurer instruction set architectures.

Some of these properties presumably belong to the non-trivial insights of practitioners involved in the design of instruction set architectures:

— Theorem 1 clarifies the ever increasing operating unit size. In principle, it is possible to undo an increase in the operating unit size originating from matters such as more advanced pipelined instruction processing, more advanced superscalar instruction issue and more accurate floating point calculations. However, the price of that is prohibitive measured by the increase in the size of the instruction set needed to produce the same state changes on the data memory concerned, the number of steps needed for those state changes, and so on.

— To a certain extent, Theorem 2 explains the existence of programming. In principle, it is possible to achieve each possible transformation on the states of the data memory of an instruction set architecture using a very short and simple program. However, that requires instructions dedicated to the different transformations and an operating unit size that is broadly the same as the data memory size.

— Theorem 3 points out the limitations imposed by existing instruction set architectures. Even if we assume the size of the operating unit, the size of the instruction set and

the maximal number of states of the threads applied are much larger than found in practice, we cannot achieve all possible transformations on the states of the data memory concerned.

Theorems 1, 2, and 3 still go through if we remove the conditions imposed on the data manipulation instructions of strict load/store Maurer instruction set architectures. These conditions originated in Bergstra and Middelburg (2008), where they were introduced to allow the exploitation of parallelism to speed up instruction processing to the maximum.

In this paper, we have taken the view that transformations on the states of the external memory are achieved by applying threads. We could have taken the less abstract view that transformations on the states of the external memory are achieved by running stored programs. This would have led to complications that appear to be unnecessary since only those threads that are represented by these programs are relevant to the transformations on the states of the external memory that are achieved. However, in the case of Theorem 3, the bounds that are imposed on strict load/store Maurer instruction set architectures and threads induce an upper bound on the number of threads that can be applied. This leads us to ask whether the number of threads that can be represented in the internal memory of the instruction set architectures concerned is possibly higher than the upper bound on the number of threads that can be applied. This question can be answered in the negative: the upper bound is the largest number of threads that can be represented in the internal memory. The upper bound is in fact much higher than the number of threads that can in practice be represented in the internal memory (*cf.* the remark following Lemma 1). This means that Theorem 3 demonstrates that load/store instruction set architectures also impose restrictions on the expressiveness of computers if we take the view that transformations on the states of the external memory are achieved by running stored programs.

One option for future work is to try to improve on the results given in the current paper. For example, we know from Theorem 2 that in order to obtain completeness with 5 data manipulation instructions and threads whose number of states is less than or equal to 8, it is sufficient to take the operating unit size to be slightly greater than the data memory size. However, we do not yet know the size of the smallest operating unit required for completeness. Another option for future work is to establish results concerning the use of half the data memory as internal memory. No such results are given in this paper. Lemma 1 assumes that half the data memory is used as internal memory because it provides a good case for the number taken as the maximal number of threads that can be applied. However, the proofs of Lemma 1 and Theorem 3 both go through without internal memory.

The speed with which transformations on the states of the external memory are achieved depends largely on the way in which the strict load/store Maurer instruction set architecture in question is implemented, which makes establishing general results about it a bit tricky. However, the speed with which transformations on the states of the external memory are achieved also depends on the volume of data transfer needed between the external memory and the operating unit. Establishing a connection between this volume and the parameters of thread powered function classes is yet another option for future work.

References

Baeten, J. C. M. and Weijland, W. P. (1990) *Process Algebra*, Cambridge Tracts in Theoretical Computer Science **18**, Cambridge University Press.

Bergstra, J. A. and Bethke, I. (2003) Polarized process algebra and program equivalence. In: Baeten, J. C. M., Lenstra, J. K., Parrow, J. and Woeginger, G. J. (eds.) Proceedings 30th ICALP. *Springer-Verlag Lecture Notes in Computer Science* **2719** 1–21.

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

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

Bergstra, J. A. and Middelburg, C. A. (2007) Maurer computers with single-thread control. *Fundamenta Informaticae* **80** (4) 333–362.

Bergstra, J. A. and Middelburg, C. A. (2008) Maurer computers for pipelined instruction processing. *Mathematical Structures in Computer Science* **18** (2) 373–409.

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

Hennessy, J. L. and Patterson, D. A. (2003) *Computer Architecture: A Quantitative Approach*, third edition, Morgan Kaufmann.

Hoare, C. A. R. (1985) *Communicating Sequential Processes*, Prentice-Hall.

Hopcroft, J. E., Motwani, R. and Ullman, J. D. (2001) *Introduction to Automata Theory, Languages and Computation*, second edition, Addison-Wesley.

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

Maurer, W. D. (2006) A theory of computer instructions. *Science of Computer Programming* **60** 244–273.

Milner, R. (1989) *Communication and Concurrency*, Prentice-Hall.

Thornton, J. (1970) *Design of a Computer – The Control Data 6600*, Scott, Foresman and Co.