Computer Science Journal of Moldova, vol.**26**, no.3(78), 2018

A Short Introduction to Program Algebra with Instructions for Boolean Registers

Jan A. Bergstra, Cornelis A. Middelburg

Abstract

A parameterized algebraic theory of instruction sequences, objects that represent the behaviours produced by instruction sequences under execution, and objects that represent the behaviours exhibited by the components of the execution environment of instruction sequences is the basis of a line of research in which issues relating to a wide variety of subjects from computer science have been rigorously investigated thinking in terms of instruction sequences. In various papers that belong to this line of research, use is made of an instantiation of this theory in which the basic instructions are instructions to read out and alter the content of Boolean registers and the components of the execution environment are Boolean registers. In this paper, we give a simplified presentation of the most general such instantiated theory.

Keywords: program algebra, thread algebra, thread-service interaction, Boolean register.

MSC 2010: 68Q05, 68Q55, 03B70.

1 Introduction

We are carrying out a line of research in which issues relating to a wide variety of subjects from computer science are rigorously investigated thinking in terms of instruction sequences (see e.g. [1]). The groundwork for this line of research is the combination of an algebraic theory of single-pass instruction sequences, called program algebra, and an algebraic theory of mathematical objects that represent the behaviours

produced by instruction sequences under execution, called basic thread algebra, extended to deal with the interaction between instruction sequences under execution and components of their execution environment (see e.g. [2]). This groundwork is parameterized by a set of basic instructions and a set of objects that represent the behaviours exhibited by the components of the execution environment.

In various papers that have resulted from this line of research, use is made of an instantiation of this theory in which certain instructions to read out and alter the content of Boolean registers are taken as basic instructions and Boolean registers are taken as the components of the execution environment (see [3]–[8]). In the current paper, we give a simplified presentation of the instantiation in which all possible instructions to read out and alter the content of Boolean registers are taken as basic instructions.

In the papers referred to above, the rationale for taking certain instructions to read out and alter the content of Boolean registers as basic instructions is that the instructions concerned are sufficient to compute each function on bit strings of any fixed length by a finite instruction sequence. However, shorter instruction sequences may be possible if certain additional instructions to read out and alter the content of Boolean registers are taken as basic instructions (see [9]). That is why we opted for the most general instantiation.

Both program algebra and basic thread algebra were first presented in [10].[1] An extension of basic thread algebra to deal with the interaction between instruction sequences under execution and components of their execution environment, called services, was presented for the first time in [11]. A substantial re-design of this extension was first presented in [12]. The presentation of both extensions is rather involved because they are parameterized and owing to this cover a generic set of basic instructions and a generic set of services. In the current paper, a much less involved presentation is obtained by covering only the case where the basic instructions are instructions to read out and alter the content of Boolean registers and the services are Boolean registers.

[1] In that paper and the first subsequent papers, basic thread algebra was introduced under the name basic polarized process algebra.

This paper is organized as follows. First, we introduce program algebra (Section 2) and basic thread algebra (Section 3) and extend their combination to make precise which behaviours are produced by instruction sequences under execution (Section 4). Next, we present the instantiation of the resulting theory in which all possible instructions to read out and alter Boolean registers are taken as basic instructions (Section 5), introduce an algebraic theory of Boolean register families (Section 6), and extend the combination of the theories presented in the two preceding sections to deal with the interaction between instruction sequences under execution and Boolean registers (Section 7). Then, we formalize in the setting of the resulting theory what it means that a given instruction sequence computes a given partial function from $\mathbb{B}^n$ to $\mathbb{B}^m$ ($n, m \in \mathbb{N}$) (Section 8) and give a survey of uses for the resulting theory (Section 9). Finally, we make some concluding remarks (Section 10).

The following should be mentioned in advance. The set $\mathbb{B}$ is a set with two elements whose intended interpretations are the truth values *false* and *true*. As is common practice, we represent the elements of $\mathbb{B}$ by the bits 0 and 1. In line with generally accepted conventions, we use terminology based on identification of the elements of $\mathbb{B}$ with their representation where appropriate. For example, the elements of $\mathbb{B}^n$ are loosely called bit strings of length n.

In this paper, some familiarity with algebraic specification is assumed. The relevant notions are explained in handbook chapters and books on algebraic specification, e.g. [13]–[16].

This paper is to a large extent a compilation of material from several earlier publications. Various examples, various explanatory remarks, and the axioms from Section 7 do not occur in earlier publications.

2 Program Algebra

In this section, we present PGA (ProGram Algebra). The starting-point of PGA is the perception of a program as a single-pass instruction sequence, i.e. a possibly infinite sequence of instructions of which each instruction is executed at most once and can be dropped after it has

been executed or jumped over. The concepts underlying the primitives of program algebra are common in programming, but the particular form of the primitives is not common. The predominant concern in the design of PGA has been to achieve simple syntax and semantics, while maintaining the expressive power of arbitrary finite control.

It is assumed that a fixed but arbitrary set $\mathcal{A}$ of *basic instructions* has been given. $\mathcal{A}$ is the basis for the set of instructions that may occur in the instruction sequences considered in PGA. The intuition is that the execution of a basic instruction may modify a state and must produce the Boolean value 0 or 1 as reply at its completion. The actual reply may be state-dependent.

In applications of PGA, the instructions taken as basic instructions vary, in effect, from instructions relating to unbounded counters, unbounded stacks or Turing tapes through instructions relating to Boolean registers or natural number registers to machine language instructions of actual computers.

The set of instructions of which the instruction sequences considered in PGA are composed is the set that consists of the following elements:

- for each $a \in \mathcal{A}$, a *plain basic instruction* a;
- for each $a \in \mathcal{A}$, a *positive test instruction* $+a$;
- for each $a \in \mathcal{A}$, a *negative test instruction* $-a$;
- for each $l \in \mathbb{N}$, a *forward jump instruction* $\#l$;
- a *termination instruction* $!$.

We write $\mathcal{I}$ for this set. The elements from this set are called *primitive instructions*.

Primitive instructions are the elements of the instruction sequences considered in PGA. On execution of such an instruction sequence, these primitive instructions have the following effects:

- the effect of a positive test instruction $+a$ is that basic instruction a is executed and execution proceeds with the next primitive

instruction if 1 is produced and otherwise the next primitive instruction is skipped and execution proceeds with the primitive instruction following the skipped one — if there is no primitive instruction to proceed with, inaction occurs;

- the effect of a negative test instruction $-a$ is the same as the effect of $+a$, but with the role of the value produced reversed;
- the effect of a plain basic instruction a is the same as the effect of $+a$, but execution always proceeds as if 1 is produced;
- the effect of a forward jump instruction $\#l$ is that execution proceeds with the lth next primitive instruction — if l equals 0 or there is no primitive instruction to proceed with, inaction occurs;
- the effect of the termination instruction ! is that execution terminates.

Inaction occurs if no more basic instructions are executed, but execution does not terminate.

A plain basic instruction a is generally used in the case of a basic instruction a that modifies a state and a positive test instruction $+a$ or a negative test instruction $-a$ is generally used in the case of a basic instruction a that does not modify a state. However, there are no rules prescribing such use.

PGA has one sort: the sort **IS** of *instruction sequences*. We make this sort explicit to anticipate the need for many-sortedness later on. To build terms of sort **IS**, PGA has the following constants and operators:

- for each $u \in \mathcal{I}$, the *instruction* constant $u :\to \mathbf{IS}$;
- the binary *concatenation* operator $\_ ; \_ : \mathbf{IS} \times \mathbf{IS} \to \mathbf{IS}$;
- the unary *repetition* operator $\_^\omega : \mathbf{IS} \to \mathbf{IS}$.

Terms of sort **IS** are built as usual in the one-sorted case. We assume that there are infinitely many variables of sort **IS**, including X, Y, Z. We use infix notation for concatenation and postfix notation for repetition. Taking these notational conventions into account, the syntax of

closed PGA terms (of sort **IS**) can be defined in Backus-Naur style as follows:

$$CT_{\mathbf{IS}} ::= a \mid +a \mid -a \mid \#l \mid ! \mid (CT_{\mathbf{IS}} \,;\, CT_{\mathbf{IS}}) \mid ({CT_{\mathbf{IS}}}^{\omega}) \;,$$

where $a \in \mathcal{A}$ and $l \in \mathbb{N}$.[2]

Throughout the paper, we generally omit grouping parentheses if they can be unambiguously added or they are unnecessary because it is axiomatized that the operator concerned stands for an associative operation.

A PGA term in which the repetition operator does not occur is called a *repetition-free* PGA term. A PGA term that is not repetition-free is said to be a PGA term that *has a repeating part*.

One way of thinking about closed PGA terms is that they represent non-empty, possibly infinite sequences of primitive instructions with finitely many distinct suffixes. The instruction sequence represented by a closed term of the form $t;t'$ is the instruction sequence represented by t concatenated with the instruction sequence represented by t'.[3] The instruction sequence represented by a closed term of the form t^{ω} is the instruction sequence represented by t concatenated infinitely many times with itself. A closed PGA term represents a finite instruction sequence if and only if it is a closed repetition-free PGA term.

A simple example of a closed PGA term is

$$(-a \,;\, (\#3 \,;\, (b \,;\, !)))^{\omega} \;.$$

On execution of the infinite instruction sequence denoted by this term, first the basic instruction a is executed repeatedly until its execution produces the reply 1, next the basic instruction b is executed, and after that execution terminates. Because $(X \,;Y) \,;Z = X \,;(Y \,;Z)$ is an axiom of PGA (see below), we could have written $(-a \,;\#3 \,;b \,;!)^{\omega}$ instead of $(-a \,;(\#3 \,;(b \,;!)))^{\omega}$ above.

The axioms of PGA are given in Table 1. In this table, u, $u_1, \ldots, u_k$

[2] We use $CT_{\mathbf{S}}$, where **S** is a sort, as nonterminal standing for closed terms of sort **S**.

[3] The concatenation of an infinite sequence with a finite or infinite sequence yields the former sequence.

Table 1. Axioms of PGA

$(X\,;Y)\,;Z = X\,;(Y\,;Z)$	PGA1
$(X^n)^\omega = X^\omega$	PGA2
$X^\omega\,;Y = X^\omega$	PGA3
$(X\,;Y)^\omega = X\,;(Y\,;X)^\omega$	PGA4
$\#k{+}1\,;u_1\,;\ldots\,;u_k\,;\#0 = \#0\,;u_1\,;\ldots\,;u_k\,;\#0$	PGA5
$\#k{+}1\,;u_1\,;\ldots\,;u_k\,;\#l = \#l{+}k{+}1\,;u_1\,;\ldots\,;u_k\,;\#l$	PGA6
$(\#l{+}k{+}1\,;u_1\,;\ldots\,;u_k)^\omega = (\#l\,;u_1\,;\ldots\,;u_k)^\omega$	PGA7
$\#l{+}k{+}k'{+}2\,;u_1\,;\ldots\,;u_k\,;(v_1\,;\ldots\,;v_{k'+1})^\omega =$ $\#l{+}k{+}1\,;u_1\,;\ldots\,;u_k\,;(v_1\,;\ldots\,;v_{k'+1})^\omega$	PGA8

and $v_1,\ldots,v_{k'+1}$ stand for arbitrary primitive instructions from $\mathcal{I}$, k, k', and l stand for arbitrary natural numbers from $\mathbb{N}$, and n stands for an arbitrary natural number from $\mathbb{N}_1$.[4] For each $n \in \mathbb{N}_1$, the term t^n, where t is a PGA term, is defined by induction on n as follows: $t^1 = t$, and $t^{n+1} = t\,;t^n$.

Some simple examples of equations derivable from the axioms of PGA are

$$(a\,;b)^\omega\,;c = a\,;(b\,;a)^\omega\ ,$$
$$+a\,;(b\,;(-c\,;\#2\,;!)^\omega)^\omega = +a\,;b\,;(-c\,;\#2\,;!)^\omega\ .$$

Closed PGA terms t and t' represent the same instruction sequence iff $t = t'$ is derivable from PGA1–PGA4. In this case, we say that the represented instruction sequences are *instruction sequence congruent*. We write $\mathrm{PGA}^{\mathrm{isc}}$ for the algebraic theory whose sorts, constants and operators are those of PGA, but whose axioms are PGA1–PGA4.

The informal explanation of closed PGA terms as sequences of primitive instructions given above can be looked upon as a sketch of the in-

[4] We write $\mathbb{N}_1$ for the set $\{n \in \mathbb{N} \mid n \geq 1\}$ of positive natural numbers.

tended model of the axioms of $\mathrm{PGA}^{\mathrm{isc}}$. This model, which is described in detail in, for example, [2], is an initial model of the axioms of $\mathrm{PGA}^{\mathrm{isc}}$.

The *unfolding equation* $X^\omega = X \,;\, X^\omega$ is derivable from the axioms of $\mathrm{PGA}^{\mathrm{isc}}$ by first taking the instance of PGA2 in which $n = 2$, then applying PGA4, and finally applying the instance of PGA2 in which $n = 2$ again.

A closed PGA term is in *first canonical form* if it is of the form t or $t \,;\, t'^\omega$, where t and t' are closed repetition-free PGA terms. The following proposition, proved in [2], relates $\mathrm{PGA}^{\mathrm{isc}}$ and first canonical forms.

Proposition 1. *For all closed* PGA *terms* t*, there exists a closed* PGA *term* t' *that is in first canonical form such that* $t = t'$ *is derivable from the axioms of* $\mathrm{PGA}^{\mathrm{isc}}$*.*

The examples given above of equations derivable from the axioms of PGA are derivable from the axioms of $\mathrm{PGA}^{\mathrm{isc}}$ only. Their left-hand sides are not in first canonical form and their right-hand sides are in first canonical form. Simple examples of equations derivable from the axioms of PGA and not derivable from the axioms of $\mathrm{PGA}^{\mathrm{isc}}$ are

$$-a \,;\, \#2 \,;\, (+b \,;\, \#2)^\omega = -a \,;\, \#0 \,;\, (+b \,;\, \#0)^\omega \;,$$
$$+a \,;\, \#6 \,;\, b \,;\, (-c \,;\, \#9)^\omega = +a \,;\, \#2 \,;\, b \,;\, (-c \,;\, \#1)^\omega \;.$$

Closed PGA terms t and t' represent the same instruction sequence after changing all chained jumps into single jumps and making all jumps as short as possible iff $t = t'$ is derivable from PGA1–PGA8. In this case, we say that the represented instruction sequences are *structurally congruent*.

A closed PGA term t *has chained jumps* if there exists a closed PGA term t' such that $t = t'$ is derivable from the axioms of $\mathrm{PGA}^{\mathrm{isc}}$ and t' contains a subterm of the form $\#n{+}1 \,;\, u_1 \,;\, \ldots \,;\, u_n \,;\, \#l$. A closed PGA term t of the form $u_1 \,;\, \ldots \,;\, u_m \,;\, (v_1 \,;\, \ldots \,;\, v_k)^\omega$ *has shortest possible jumps* if: (i) for each $i \in [1, m]$ for which u_i is of the form $\#l$, $l \leq k + m - i$; (ii) for each $j \in [1, k]$ for which v_j is of the form $\#l$, $l \leq k - 1$. A closed PGA term is in *second canonical form* if it is in first canonical form,

does not have chained jumps, and has shortest possible jumps if it has a repeating part. The following proposition, proved in [2], relates PGA and second canonical forms.

Proposition 2. *For all closed* PGA *terms* t*, there exists a closed* PGA *term* t' *that is in second canonical form such that* $t = t'$ *is derivable from the axioms of* PGA*.*

The examples given above of equations derivable from the axioms of PGA and not derivable from the axioms of $\mathrm{PGA}^{\mathrm{isc}}$ have left-hand sides that are not in second canonical form and right-hand sides that are in second canonical form.

Henceforth, the instruction sequences of the kind considered in PGA are called PGA instruction sequences.

In Section 7, we will use the notation $;_{i=1}^{n} t_i$. For each $i \in \mathbb{N}_1$, let t_i be PGA terms. Then, for each $n \in \mathbb{N}_1$, the term $;_{i=1}^{n} t_i$ is defined by induction on n as follows: $;_{i=1}^{1} t_i = t_1$ and $;_{i=1}^{n+1} t_i = ;_{i=1}^{n} t_i \,;\, t_{n+1}$.

3 Basic Thread Algebra for Finite and Infinite Threads

In this section, we present BTA (Basic Thread Algebra) and an extension of BTA that reflects the idea that infinite threads are identical if their approximations up to any finite depth are identical.

BTA is concerned with mathematical objects that model in a direct way the behaviours produced by PGA instruction sequences under execution. The objects in question are called threads. A thread models a behaviour that consists of performing basic actions in a sequential fashion. Upon performing a basic action, a reply from an execution environment determines how the behaviour proceeds subsequently. The possible replies are the Boolean values 0 and 1.

The basic instructions from $\mathcal{A}$ are taken as basic actions. Besides, tau is taken as a special basic action. It is assumed that $\mathsf{tau} \notin \mathcal{A}$. We write $\mathcal{A}_{\mathsf{tau}}$ for $\mathcal{A} \cup \{\mathsf{tau}\}$.

BTA has one sort: the sort $\mathbf{T}$ of *threads*. We make this sort explicit to anticipate the need for many-sortedness later on. To build terms of sort $\mathbf{T}$, BTA has the following constants and operators:

- the *inaction* constant $\mathsf{D} :\to \mathbf{T}$;
- the *termination* constant $\mathsf{S} :\to \mathbf{T}$;
- for each $\alpha \in \mathcal{A}_{\mathsf{tau}}$, the binary *postconditional composition* operator $\_ \trianglelefteq \alpha \trianglerighteq \_ : \mathbf{T} \times \mathbf{T} \to \mathbf{T}$.

Terms of sort $\mathbf{T}$ are built as usual in the one-sorted case. We assume that there are infinitely many variables of sort $\mathbf{T}$, including x, y, z. We use infix notation for postconditional composition. Taking this notational convention into account, the syntax of closed BTA terms (of sort $\mathbf{T}$) can be defined in Backus-Naur style as follows:

$$CT_{\mathbf{T}} ::= \mathsf{D} \mid \mathsf{S} \mid (CT_{\mathbf{T}} \trianglelefteq \alpha \trianglerighteq CT_{\mathbf{T}}) \,,$$

where $\alpha \in \mathcal{A}_{\mathsf{tau}}$. We introduce *basic action prefixing* as an abbreviation: $\alpha \circ t$, where $\alpha \in \mathcal{A}_{\mathsf{tau}}$ and t is a BTA term, abbreviates $t \trianglelefteq \alpha \trianglerighteq t$. We treat an expression of the form $\alpha \circ t$ and the BTA term that it abbreviates as syntactically the same.

Closed BTA terms are considered to represent threads. The thread represented by a closed term of the form $t \trianglelefteq \alpha \trianglerighteq t'$ models the behaviour that first performs α, and then proceeds as the behaviour modeled by the thread represented by t if the reply from the execution environment is 1 and proceeds as the behaviour modeled by the thread represented by t' if the reply from the execution environment is 0. Performing tau, which is considered performing an internal action, always leads to the reply 1. The thread represented by S models the behaviour that does nothing else but terminate and the thread represented by D models the behaviour that is inactive, i.e. it performs no more basic actions and it does not terminate.

A simple example of a closed BTA term is

$$(b \circ \mathsf{S}) \trianglelefteq a \trianglerighteq \mathsf{D} \,.$$

Table 2. Axioms of BTA

$x \trianglelefteq \mathsf{tau} \trianglerighteq y = x \trianglelefteq \mathsf{tau} \trianglerighteq x$	T1

This term denotes the thread that first performs basic action a, if the reply from the execution environment on performing a is 1, it next performs the basic action b and then terminates, and if the reply from the execution environment on performing a is 0, it next becomes inactive.

BTA has only one axiom. This axiom is given in Table 2. Using the abbreviation introduced above, it can also be written as follows: $x \trianglelefteq \mathsf{tau} \trianglerighteq y = \mathsf{tau} \circ x$.

Each closed BTA term represents a finite thread, i.e. a thread with a finite upper bound to the number of basic actions that it can perform. Infinite threads, i.e. threads without a finite upper bound to the number of basic actions that it can perform, can be defined by means of a set of recursion equations (see e.g. [12]).

A simple example of a set of recursion equations that consists of a single equation is

$$x = (b \circ \mathsf{S}) \trianglelefteq a \trianglerighteq x \ .$$

Its solution is the thread that first repeatedly performs basic action a until the reply from the execution environment on performing a is 1, next performs the basic action b and then terminates.

A regular thread is a finite or infinite thread that can be defined by means of a finite set of recursion equations. The behaviours produced by PGA instruction sequences under execution are exactly the behaviours modeled by regular threads.

Two infinite threads are considered identical if their approximations up to any finite depth are identical. The approximation up to depth n of a thread models the behaviour that differs from the behaviour modeled by the thread in that it will become inactive after it has performed n actions unless it would terminate at this point. AIP (Approximation Induction Principle) is a conditional equation that formalizes the above-mentioned view on infinite threads. In AIP, the approximation

Table 3. Axioms for the projection operators and AIP

$\pi_0(x) = \mathsf{D}$	PR1
$\pi_{n+1}(\mathsf{D}) = \mathsf{D}$	PR2
$\pi_{n+1}(\mathsf{S}) = \mathsf{S}$	PR3
$\pi_{n+1}(x \trianglelefteq \alpha \trianglerighteq y) = \pi_n(x) \trianglelefteq \alpha \trianglerighteq \pi_n(y)$	PR4
$\bigwedge_{n \geq 0} \pi_n(x) = \pi_n(y) \Rightarrow x = y$	AIP

up to depth n is phrased in terms of the unary *projection* operator $\pi_n : \mathbf{T} \to \mathbf{T}$.

The axioms for the projection operators and AIP are given in Table 3. In this table, α stands for an arbitrary basic action from $\mathcal{A}_{\mathsf{tau}}$ and n stands for an arbitrary natural number from $\mathbb{N}$. We write BTA$^\infty$ for BTA extended with the projection operators, the axioms for the projection operators, and AIP.

By AIP, we have to deal in BTA$^\infty$ with conditional equational formulas with a countably infinite number of premises. Therefore, infinitary conditional equational logic is used in deriving equations from the axioms of BTA$^\infty$. A complete inference system for infinitary conditional equational logic can be found in, for example, [17].

For a simple example of the use of the axioms for the projection operators and AIP, we consider the (recursion) equations $x = a \circ x$ and $y = a \circ a \circ y$. With these equations as hypotheses, the following equations are derivable from the axioms for the projection operators:

$$\begin{aligned}
&\pi_0(x) = \mathsf{D}\,, && \pi_0(y) = \mathsf{D}\,,\\
&\pi_1(x) = a \circ \mathsf{D}\,, && \pi_1(y) = a \circ \mathsf{D}\,,\\
&\pi_2(x) = a \circ a \circ \mathsf{D}\,, && \pi_2(y) = a \circ a \circ \mathsf{D}\,,\\
&\pi_3(x) = a \circ a \circ a \circ \mathsf{D}\,, && \pi_3(y) = a \circ a \circ a \circ \mathsf{D}\,,\\
&\qquad\vdots
\end{aligned}$$

Hence, the conditional equation $x = a \circ x \wedge y = a \circ a \circ y \Rightarrow x = y$ is

derivable from the axioms for the projection operators and AIP. This conditional equation tells us that the recursion equations $x = a \circ x$ and $y = a \circ a \circ y$ have the same solution.

4 Thread Extraction and Behavioural Congruence

In this section, we make precise in the setting of BTA^∞ which behaviours are produced by PGA instruction sequences under execution and introduce the notion of behavioural congruence on PGA instruction sequences.

To make precise which behaviours are produced by PGA instruction sequences under execution, we introduce an operator $|\_|$ meant for extracting from each PGA instruction sequence the thread that models the behaviour produced by it under execution. For each closed PGA term t, $|t|$ represents the thread that models the behaviour produced by the instruction sequence represented by t under execution.

Formally, we combine PGA with BTA^∞ and extend the combination with the *thread extraction* operator $|\_| : \mathbf{IS} \to \mathbf{T}$ and the axioms given in Table 4. In this table, a stands for an arbitrary basic instruction from $\mathcal{A}$, u stands for an arbitrary primitive instruction from $\mathcal{I}$, and l stands for an arbitrary natural number from $\mathbb{N}$. We write

Table 4. Axioms for the thread extraction operator

$\lvert a\rvert = a \circ \mathsf{D}$	TE1	$\lvert \#l\rvert = \mathsf{D}$	TE7
$\lvert a\,;X\rvert = a \circ \lvert X\rvert$	TE2	$\lvert \#0\,;X\rvert = \mathsf{D}$	TE8
$\lvert +a\rvert = a \circ \mathsf{D}$	TE3	$\lvert \#1\,;X\rvert = \lvert X\rvert$	TE9
$\lvert +a\,;X\rvert = \lvert X\rvert \trianglelefteq a \trianglerighteq \lvert \#2\,;X\rvert$	TE4	$\lvert \#l+2\,;u\rvert = \mathsf{D}$	TE10
$\lvert -a\rvert = a \circ \mathsf{D}$	TE5	$\lvert \#l+2\,;u\,;X\rvert = \lvert \#l+1\,;X\rvert$	TE11
$\lvert -a\,;X\rvert = \lvert \#2\,;X\rvert \trianglelefteq a \trianglerighteq \lvert X\rvert$	TE6	$\lvert !\rvert = \mathsf{S}$	TE12
		$\lvert !\,;X\rvert = \mathsf{S}$	TE13

PGA/BTA$^\infty$ for the combination of PGA and BTA$^\infty$ extended with the thread extraction operator and the axioms for the thread extraction operator. The syntax of closed PGA/BTA$^\infty$ terms of sort $\mathbf{T}$ can be defined in Backus-Naur style as follows:

$$CT'_{\mathbf{T}} ::= \mathsf{D} \mid \mathsf{S} \mid (CT'_{\mathbf{T}} \trianglelefteq \alpha \trianglerighteq CT'_{\mathbf{T}}) \mid |CT_{\mathbf{IS}}| \;,$$

where $\alpha \in \mathcal{A}_{\mathsf{tau}}$. $CT_{\mathbf{IS}}$ is defined in Section 2.

A simple example of thread extraction is

$$|{+}a \,;\, \#2 \,;\, \#3 \,;\, b \,;\, !| = (b \circ \mathsf{S}) \trianglelefteq a \trianglerighteq \mathsf{D} \;.$$

In the case of infinite instruction sequences, thread extraction yields threads definable by means of a set of recursion equations. For example,

$$|({+}a \,;\, \#2 \,;\, \#3 \,;\, b \,;\, !)^\omega|$$

is the solution of the set of recursion equations that consists of the single equation

$$x = (b \circ \mathsf{S}) \trianglelefteq a \trianglerighteq x \;.$$

If a closed PGA term t represents an instruction sequence that starts with an infinite chain of forward jumps, then TE9 and TE11 can be applied to $|t|$ infinitely often without ever showing that a basic action is performed. In this case, we have to do with inaction and, being consistent with that, $|t| = \mathsf{D}$ is derivable from the axioms of PGA and TE1–TE13. By contrast, $|t| = \mathsf{D}$ is not derivable from the axioms of PGA$^{\mathrm{isc}}$ and TE1–TE13. However, if closed PGA terms t and t' represent instruction sequences in which no infinite chains of forward jumps occur, then $t = t'$ is derivable from the axioms of PGA only if $|t| = |t'|$ is derivable from the axioms of PGA$^{\mathrm{isc}}$ and TE1–TE13.

If a closed PGA term t represents an infinite instruction sequence, then we can extract the approximations of the thread modeling the behaviour produced by that instruction sequence under execution up to every finite depth: for each $n \in \mathbb{N}$, there exists a closed BTA term t'' such that $\pi_n(|t|) = t''$ is derivable from the axioms of PGA, TE1–TE13, the axioms of BTA, and PR1–PR4. If closed PGA terms t

and t' represent infinite instruction sequences that produce the same behaviour under execution, then this can be proved using the following instance of AIP: $\bigwedge_{n\geq 0} \pi_n(|t|) = \pi_n(|t'|) \Rightarrow |t| = |t'|$.

PGA instruction sequences are behaviourally equivalent if they produce the same behaviour under execution. Behavioural equivalence is not a congruence. Instruction sequences are behaviourally congruent if they produce the same behaviour irrespective of the way they are entered and the way they are left.

Let t and t' be closed PGA terms. Then:

- t and t' are *behaviourally equivalent*, written $t \equiv_{\mathsf{be}} t'$, if $|t| = |t'|$ is derivable from the axioms of PGA/BTA$^\infty$.
- t and t' are *behaviourally congruent*, written $t \cong_{\mathsf{bc}} t'$, if, for each $l, n \in \mathbb{N}$, $\#l \,;\, t \,;\, !^n \equiv_{\mathsf{be}} \#l \,;\, t' \,;\, !^n$.[5]

Some simple examples of behavioural equivalence are

$$a \,;\, \#2 \,;\, +b \,;\, ! \equiv_{\mathsf{be}} a \,;\, \#2 \,;\, +c \,;\, ! \;,$$
$$(+a \,;\, \#2 \,;\, \#3 \,;\, b \,;\, !)^\omega \equiv_{\mathsf{be}} (-a \,;\, \#3 \,;\, b \,;\, !)^\omega \;.$$

We cannot lift these examples to behavioural congruence, i.e.

$$a \,;\, \#2 \,;\, +b \,;\, ! \not\cong_{\mathsf{bc}} a \,;\, \#2 \,;\, +c \,;\, ! \;,$$
$$(+a \,;\, \#2 \,;\, \#3 \,;\, b \,;\, !)^\omega \not\cong_{\mathsf{bc}} (-a \,;\, \#3 \,;\, b \,;\, !)^\omega \;.$$

A simple example of behavioural congruence is

$$(+a \,;\, \#3 \,;\, \#2 \,;\, b)^\omega \cong_{\mathsf{bc}} (-a \,;\, \#3 \,;\, \#2 \,;\, b)^\omega \;.$$

It is proved in [2] that each closed PGA term is behaviourally equivalent to a term of the form t^ω, where t is a closed repetition-free PGA term.

Proposition 3. *For all closed* PGA *terms t, there exists a closed repetition-free* PGA *term t' such that $t \equiv_{\mathsf{be}} t'^\omega$.*

[5] We use the convention that $t \,;\, t'^0$ stands for t.

Behavioural congruence is the largest congruence contained in behavioural equivalence. Moreover, structural congruence implies behavioural congruence.

Proposition 4. *For all closed* PGA *terms* t *and* t', $t = t'$ *is derivable from the axioms of* PGA *only if* $t \mathrel{\mathsf{bc}} t'$.

Proof. The proof is basically the proof of Proposition 2.2 from [2]. In that proof use is made of the uniqueness of solutions of sets of recursion equations where each right-hand side is a BTA term of the form D, S or $s \trianglelefteq \alpha \trianglerighteq s'$ with BTA terms s and s' that contain only variables occurring as one of the right-hand sides. This uniqueness follows from AIP (see also Corollary 2.1 from [2]). □

Conversely, behavioural congruence does not imply structural congruence. For example, $+a\,;!\,;!\mathrel{\mathsf{bc}} -a\,;!\,;!$, but $+a\,;!\,;! = -a\,;!\,;!$ is not derivable from the axioms of PGA.

In [18], we present an equational axiom system for behavioural congruence that is sound for closed PGA terms and complete for closed repetition-free PGA terms.

The following proposition, proved in [2], puts the expressiveness of PGA in terms of producible behaviours.

Proposition 5. *Let* $\mathcal{M}$ *be a model of* PGA/BTA$^\infty$. *Then, for each element* p *from the domain associated with the sort* $\mathbf{T}$ *in* $\mathcal{M}$, *there exists a closed* PGA *term* t *such that* p *is the interpretation of* $|t|$ *in* $\mathcal{M}$ *iff* p *is a component of the solution of a finite set of recursion equations* $\{V = t_V \mid V \in \mathcal{V}\}$, *where* $\mathcal{V}$ *is a set of variables of sort* $\mathbf{T}$ *and each* t_V *is a* BTA *term that is not a variable and contains only variables from* $\mathcal{V}$.

More results on the expressiveness of PGA can be found in [2].

5 The Case of Instructions for Boolean Registers

In this section, we present the instantiation of PGA in which all possible instructions to read out and alter Boolean registers are taken as basic

instructions.

In this instantiation, it is assumed that a fixed but arbitrary set $\mathcal{F}$ of *foci* has been given. Foci serve as names of Boolean registers.

The set of basic instructions used in this instantiation consists of the following:

- for each $f \in \mathcal{F}$ and $p, q : \mathbb{B} \to \mathbb{B}$, a *basic Boolean register instruction* $f.p/q$.

We write $\mathcal{A}_{\mathrm{br}}$ for this set.

Each basic Boolean register instruction consists of two parts separated by a dot. The part on the left-hand side of the dot plays the role of the name of a Boolean register and the part on the right-hand side of the dot plays the role of an operation to be carried out on the named Boolean register when the instruction is executed. The intuition is basically that carrying out the operation concerned modifies the content of the named Boolean register and produces as a reply a Boolean value that depends on the content of the named Boolean register. More precisely, the execution of a basic Boolean register instruction $f.p/q$ has the following effects:

- if the content of the Boolean register named f is b when the execution of $f.p/q$ starts, then its content is $q(b)$ when the execution of $f.p/q$ terminates;
- if the content of the Boolean register named f is b when the execution of $f.p/q$ starts, then the reply produced on termination of the execution of $f.p/q$ is $p(b)$.

The execution of $f.p/q$ has no effect on the content of Boolean registers other than the one named f.

$\mathbb{B} \to \mathbb{B}$, the set of all unary Boolean functions, consists of the following four functions:

- the function 0, satisfying $0(0) = 0$ and $0(1) = 0$;
- the function 1, satisfying $1(0) = 1$ and $1(1) = 1$;

- the function i, satisfying $\mathrm{i}(0) = 0$ and $\mathrm{i}(1) = 1$;
- the function c, satisfying $\mathrm{c}(0) = 1$ and $\mathrm{c}(1) = 0$.

In [3]–[5],[7],[8], we actually used the operations $0/0$, $1/1$, and i/i, but denoted them by $\mathsf{set}{:}0$, $\mathsf{set}{:}1$ and get, respectively. In [6], we actually used, in addition to these operations, the operation c/c, but denoted it by com. Two examples of peculiar operations are $0/\mathrm{i}$ and $1/\mathrm{i}$. Carrying out one of these operations on a Boolean register does not modify the content of the Boolean register and produces as a reply, irrespective of the content of the Boolean register, always the same Boolean value.

We write $[\mathrm{PGA/BTA}^\infty](\mathcal{A}_{\mathrm{br}})$ for $\mathrm{PGA/BTA}^\infty$ with $\mathcal{A}$ instantiated by $\mathcal{A}_{\mathrm{br}}$. Notice that $[\mathrm{PGA/BTA}^\infty](\mathcal{A}_{\mathrm{br}})$ is itself parameterized by a set of foci.

In the papers just mentioned, $\mathcal{F}$ is instantiated by

$$\{\mathsf{in}{:}i \mid i \in \mathbb{N}_1\} \cup \{\mathsf{out}{:}i \mid i \in \mathbb{N}_1\} \cup \{\mathsf{aux}{:}i \mid i \in \mathbb{N}_1\}$$

if the computation of functions from $\mathbb{B}^n$ to $\mathbb{B}^m$ with $m > 1$ is in order and

$$\{\mathsf{in}{:}i \mid i \in \mathbb{N}_1\} \cup \{\mathsf{out}\} \cup \{\mathsf{aux}{:}i \mid i \in \mathbb{N}_1\}$$

if only the computation of functions from $\mathbb{B}^n$ to $\mathbb{B}$ is in order. These foci are employed as follows:

- the foci of the form $\mathsf{in}{:}i$ serve as names of Boolean registers that are used as input registers in instruction sequences;
- the foci of the form $\mathsf{out}{:}i$ and out serve as names of Boolean registers that are used as output registers in instruction sequences;
- the foci of the form $\mathsf{aux}{:}i$ serve as names of Boolean registers that are used as auxiliary registers in instruction sequences.

The above sets of foci are just examples of sets by which $\mathcal{F}$ may be instantiated. In the algebraic theories presented in Sections 6 and 7, $\mathcal{F}$ is not instantiated.

6 Boolean Register Families

PGA instruction sequences under execution may interact with the named Boolean registers from a family of Boolean registers provided by their execution environment. In this section, we introduce an algebraic theory of Boolean register families called BRFA (Boolean Register Family Algebra). Boolean register families are reminiscent of the Boolean register files found in the central processing unit of a computer (see e.g. [19]).

In BRFA, as in [PGA/BTA$^\infty$]($\mathcal{A}_{\mathrm{br}}$), it is assumed that a fixed but arbitrary set $\mathcal{F}$ of foci has been given.

BRFA has one sort: the sort **BRF** of *Boolean register families*. To build terms of sort **BRF**, BRFA has the following constants and operators:

- the *empty Boolean register family* constant $\emptyset :\to \mathbf{BRF}$;
- for each $f \in \mathcal{F}$ and $b \in \mathbb{B} \cup \{*\}$, the *singleton Boolean register family* constant $f.\mathsf{br}(b) :\to \mathbf{BRF}$;
- the binary *Boolean register family composition* operator $\_ \oplus \_ : \mathbf{BRF} \times \mathbf{BRF} \to \mathbf{BRF}$;
- for each $F \subseteq \mathcal{F}$, the unary *encapsulation* operator $\partial_F : \mathbf{BRF} \to \mathbf{BRF}$.

We assume that there are infinitely many variables of sort **BRF**, including u, v, w. We use infix notation for the Boolean register family composition operator. Taking this notational convention into account, the syntax of closed BRFA terms (of sort **BRF**) can be defined in Backus-Naur style as follows:

$$CT_{\mathbf{BRF}} ::= \emptyset \mid f.\mathsf{br}(b) \mid (CT_{\mathbf{BRF}} \oplus CT_{\mathbf{BRF}}) \mid \partial_F(CT_{\mathbf{BRF}}) \,,$$

where $f \in \mathcal{F}$, $b \in \mathbb{B} \cup \{*\}$, and $F \subseteq \mathcal{F}$.

The Boolean register family denoted by $\emptyset$ is the empty Boolean register family. The Boolean register family denoted by a closed term

of the form $f.\mathsf{br}(b)$, where $b \in \mathbb{B}$, consists of one named Boolean register only, the Boolean register concerned is an operative Boolean register named f whose content is b. The Boolean register family denoted by a closed term of the form $f.\mathsf{br}(*)$ consists of one named Boolean register only, the Boolean register concerned is an inoperative Boolean register named f. The Boolean register family denoted by a closed term of the form $t \oplus t'$ consists of all named Boolean registers that belong to either the Boolean register family denoted by t or the Boolean register family denoted by t'. In the case where a named Boolean register from the Boolean register family denoted by t and a named Boolean register from the Boolean register family denoted by t' have the same name, they collapse to an inoperative Boolean register with the name concerned. The Boolean register family denoted by a closed term of the form $\partial_F(t)$ consists of all named Boolean registers with a name not in F that belong to the Boolean register family denoted by t.

A simple example of a Boolean register family is

$$\begin{aligned}&\mathsf{aux}{:}8.\mathsf{br}(1) \oplus \mathsf{aux}{:}7.\mathsf{br}(1) \oplus \mathsf{aux}{:}6.\mathsf{br}(0) \oplus \mathsf{aux}{:}5.\mathsf{br}(0)\\ &\quad \oplus \mathsf{aux}{:}4.\mathsf{br}(1) \oplus \mathsf{aux}{:}3.\mathsf{br}(1) \oplus \mathsf{aux}{:}2.\mathsf{br}(1) \oplus \mathsf{aux}{:}1.\mathsf{br}(0)\ .\end{aligned}$$

This Boolean register family can be seen as a storage cell whose content is the bit string 01110011. Taking the content of such storage cells for binary representations of natural numbers, the functions on bit strings of length 8 that model addition, subtraction, and multiplication modulo 2^8 of natural numbers less than 2^8 can be computed using the instructions for Boolean registers introduced in Section 5.

An inoperative Boolean register can be viewed as a Boolean register whose content is unavailable. Carrying out an operation on an inoperative Boolean register is impossible.

The axioms of BRFA are given in Table 5. In this table, f stands for an arbitrary focus from $\mathcal{F}$, F stands for an arbitrary subset of $\mathcal{F}$, and b and b' stand for arbitrary values from $\mathbb{B} \cup \{*\}$. These axioms simply formalize the informal explanation given above.

The following two propositions, proved in [2], concern an elimination result and a representation result for closed BRFA terms.

Table 5. Axioms of BRFA

$u \oplus \emptyset = u$	BRFC1	$\partial_F(\emptyset) = \emptyset$		BRFE1
$u \oplus v = v \oplus u$	BRFC2	$\partial_F(f.\mathsf{br}(b)) = \emptyset$	if $f \in F$	BRFE2
$(u \oplus v) \oplus w = u \oplus (v \oplus w)$	BRFC3	$\partial_F(f.\mathsf{br}(b)) = f.\mathsf{br}(b)$	if $f \notin F$	BRFE3
$f.\mathsf{br}(b) \oplus f.\mathsf{br}(b') = f.\mathsf{br}(*)$	BRFC4	$\partial_F(u \oplus v) = \partial_F(u) \oplus \partial_F(v)$		BRFE4

Proposition 6. *For all closed* BRFA *terms t, there exists a closed* BRFA *term t' in which encapsulation operators do not occur such that $t = t'$ is derivable from the axioms of* BRFA.

Proposition 7. *For all closed* BRFA *terms t, for all $f \in \mathcal{F}$, either $t = \partial_{\{f\}}(t)$ is derivable from the axioms of* BRFA *or there exists a $b \in \mathbb{B} \cup \{*\}$ such that $t = f.\mathsf{br}(b) \oplus \partial_{\{f\}}(t)$ is derivable from the axioms of* BRFA.

In Section 8, we will use the notation $\bigoplus_{i=1}^{n} t_i$. For each $i \in \mathbb{N}_1$, let t_i be a term of sort **BRF**. Then, for each $n \in \mathbb{N}_1$, the term $\bigoplus_{i=1}^{n} t_i$ is defined by induction on n as follows: $\bigoplus_{i=1}^{1} t_i = t_1$ and $\bigoplus_{i=1}^{n+1} t_i = \bigoplus_{i=1}^{n} t_i \oplus t_{n+1}$.

7 Interaction of Threads with Boolean Registers

A PGA instruction sequence under execution may interact with the named Boolean registers from the family of Boolean registers provided by its execution environment. In line with this kind of interaction, a thread may perform a basic action basically for the purpose of modifying the content of a named Boolean register or receiving a reply value that depends on the content of a named Boolean register. In this section, we introduce related operators.

We combine PGA/BTA$^{\infty}(\mathcal{A}_{\mathrm{br}})$ with BRFA and extend the combination with the following operators for interaction of threads with Boolean registers:

- the binary *use* operator $\_ \,/\, \_ : \mathbf{T} \times \mathbf{BRF} \to \mathbf{T}$;
- the binary *apply* operator $\_ \bullet \_ : \mathbf{T} \times \mathbf{BRF} \to \mathbf{BRF}$;
- the unary *abstraction* operator $\tau_{\mathsf{tau}} : \mathbf{T} \to \mathbf{T}$;

and the axioms given in Tables 6.[6] In these tables, f stands for an arbitrary focus from $\mathcal{F}$, p and q stand for arbitrary Boolean functions from $\mathbb{B} \to \mathbb{B}$, b stands for an arbitrary Boolean value from $\mathbb{B}$, n stands for an arbitrary natural number from $\mathbb{N}$, and t and s stand for arbitrary terms of sort **BRF**. We use infix notation for the use and apply operators. We write [PGA/BTA$^\infty$]($\mathcal{A}_{\mathrm{br}}$)/BRI for the combination of [PGA/BTA$^\infty$]($\mathcal{A}_{\mathrm{br}}$) and BRFA extended with the use operator, the apply operator, the abstraction operator, and the axioms for these operators. The syntax of closed [PGA/BTA$^\infty$]($\mathcal{A}_{\mathrm{br}}$)/BRI terms of sort **T** and **BRF** can be defined in Backus-Naur style as follows:

$$\begin{aligned} CT''_{\mathbf{T}} ::= {} & \mathsf{D} \mid \mathsf{S} \mid (CT''_{\mathbf{T}} \trianglelefteq \alpha \trianglerighteq CT''_{\mathbf{T}}) \mid |CT_{\mathbf{IS}}| \\ & \mid (CT''_{\mathbf{T}} \,/\, CT'_{\mathbf{BRF}}) \mid \tau_{\mathsf{tau}}(CT''_{\mathbf{T}}) \,, \end{aligned}$$

$$\begin{aligned} CT'_{\mathbf{BRF}} ::= {} & \emptyset \mid f.\mathsf{br}(b) \mid (CT'_{\mathbf{BRF}} \oplus CT'_{\mathbf{BRF}}) \mid \partial_F(CT'_{\mathbf{BRF}}) \\ & \mid (CT''_{\mathbf{T}} \bullet CT'_{\mathbf{BRF}}) \,, \end{aligned}$$

where $\alpha \in \mathcal{A}_{\mathrm{br}} \cup \{\mathsf{tau}\}$, $f \in \mathcal{F}$, $b \in \mathbb{B} \cup \{*\}$, $F \subseteq \mathcal{F}$. $CT_{\mathbf{IS}}$ is defined in Section 2.

Axioms U1–U7 and A1–A7 formalize the informal explanation of the use operator and the apply operator given below and in addition stipulate what is the result of apply if an unavailable focus is involved (A4) and what is the result of use and apply if an inoperative Boolean register is involved (U7 and A7). Axioms U8 and A8 allow of reasoning about infinite threads, and therefore about the behaviour produced by infinite instruction sequences under execution, in the context of use and apply, respectively.

On interaction between a thread and a Boolean register, the thread affects the Boolean register and the Boolean register affects the thread.

[6] We write $t[t'/x]$ for the result of substituting term t' for variable x in term t.

Table 6. Axioms for the use, apply and abstraction operator

$\mathsf{S} \mathbin{/} u = \mathsf{S}$	U1
$\mathsf{D} \mathbin{/} u = \mathsf{D}$	U2
$(\mathsf{tau} \circ x) \mathbin{/} u = \mathsf{tau} \circ (x \mathbin{/} u)$	U3
$(x \trianglelefteq f.p/q \trianglerighteq y) \mathbin{/} \partial_{\{f\}}(u) = (x \mathbin{/} \partial_{\{f\}}(u)) \trianglelefteq f.p/q \trianglerighteq (y \mathbin{/} \partial_{\{f\}}(u))$	U4
$(x \trianglelefteq f.p/q \trianglerighteq y) \mathbin{/} (f.\mathsf{br}(b) \oplus \partial_{\{f\}}(u)) = \mathsf{tau} \circ (x \mathbin{/} (f.\mathsf{br}(q(b)) \oplus \partial_{\{f\}}(u)))$ if $p(b) = 1$	U5
$(x \trianglelefteq f.p/q \trianglerighteq y) \mathbin{/} (f.\mathsf{br}(b) \oplus \partial_{\{f\}}(u)) = \mathsf{tau} \circ (y \mathbin{/} (f.\mathsf{br}(q(b)) \oplus \partial_{\{f\}}(u)))$ if $p(b) = 0$	U6
$(x \trianglelefteq f.p/q \trianglerighteq y) \mathbin{/} (f.\mathsf{br}(*) \oplus \partial_{\{f\}}(u)) = \mathsf{D}$	U7
$\pi_n(x \mathbin{/} u) = \pi_n(x) \mathbin{/} u$	U8
$\mathsf{S} \bullet u = u$	A1
$\mathsf{D} \bullet u = \emptyset$	A2
$(\mathsf{tau} \circ x) \bullet u = \mathsf{tau} \circ (x \bullet u)$	A3
$(x \trianglelefteq f.p/q \trianglerighteq y) \bullet \partial_{\{f\}}(u) = \emptyset$	A4
$(x \trianglelefteq f.p/q \trianglerighteq y) \bullet (f.\mathsf{br}(b) \oplus \partial_{\{f\}}(u)) = x \bullet (f.\mathsf{br}(q(b)) \oplus \partial_{\{f\}}(u))$ if $p(b) = 1$	A5
$(x \trianglelefteq f.p/q \trianglerighteq y) \bullet (f.\mathsf{br}(b) \oplus \partial_{\{f\}}(u)) = y \bullet (f.\mathsf{br}(q(b)) \oplus \partial_{\{f\}}(u))$ if $p(b) = 0$	A6
$(x \trianglelefteq f.p/q \trianglerighteq y) \bullet (f.\mathsf{br}(*) \oplus \partial_{\{f\}}(u)) = \emptyset$	A7
$\bigwedge_{k \geq n} t[\pi_k(x)/z] = s[\pi_k(y)/z] \Rightarrow t[x/z] = s[y/z]$	A8
$\tau_{\mathsf{tau}}(\mathsf{S}) = \mathsf{S}$	C1
$\tau_{\mathsf{tau}}(\mathsf{D}) = \mathsf{D}$	C2
$\tau_{\mathsf{tau}}(\mathsf{tau} \circ x) = \tau_{\mathsf{tau}}(x)$	C3
$\tau_{\mathsf{tau}}(x \trianglelefteq f.p/q \trianglerighteq y) = \tau_{\mathsf{tau}}(x) \trianglelefteq f.p/q \trianglerighteq \tau_{\mathsf{tau}}(y)$	C4
$\bigwedge_{n \geq 0} \tau_{\mathsf{tau}}(\pi_n(x)) = \tau_{\mathsf{tau}}(\pi_n(y)) \Rightarrow \tau_{\mathsf{tau}}(x) = \tau_{\mathsf{tau}}(y)$	C5

The use operator concerns the effects of Boolean registers on threads and the apply operator concerns the effects of threads on Boolean registers. The thread denoted by a closed term of the form $t \mathbin{/} t'$ and the Boolean register family denoted by a closed term of the form $t \bullet t'$ are the thread and Boolean register family, respectively, that result from carrying out the operation that is part of each basic action performed by the thread denoted by t on the Boolean register in the Boolean register family denoted by t' with the focus that is part of the basic action as its name. When the operation that is part of a basic action performed by a thread is carried out on a Boolean register, the content of the Boolean register is modified according to the operation concerned and the thread is affected as follows: the basic action turns into the internal action tau and the two ways to proceed reduce to one on the basis of the reply value produced according to the operation concerned.

With the use operator the internal action tau is left as a trace of each basic action that has led to carrying out an operation on a Boolean register. The abstraction operator serves to abstract fully from such internal activity by concealing tau. Axioms C1–C4 formalize the concealment of tau. Axiom C5 allows of reasoning about infinite threads in the context of abstraction.

A simple example of use and apply is

$$\begin{aligned}
&|\,;_{i=1}^{4}(-\mathsf{aux}{:}i.\mathrm{i}/\mathrm{i}\,;\#3\,;\mathsf{aux}{:}i.0/0\,;!\,;\mathsf{aux}{:}i.1/1)| \\
&\quad /\ \mathsf{aux}{:}4.\mathsf{br}(1) \oplus \mathsf{aux}{:}3.\mathsf{br}(1) \oplus \mathsf{aux}{:}2.\mathsf{br}(1) \oplus \mathsf{aux}{:}1.\mathsf{br}(0) \\
&= \mathsf{tau} \circ \mathsf{tau} \circ \mathsf{tau} \circ \mathsf{tau} \circ \mathsf{S}\ ,
\end{aligned}$$

$$\begin{aligned}
&|\,;_{i=1}^{4}(-\mathsf{aux}{:}i.\mathrm{i}/\mathrm{i}\,;\#3\,;\mathsf{aux}{:}i.0/0\,;!\,;\mathsf{aux}{:}i.1/1)| \\
&\quad \bullet\ \mathsf{aux}{:}4.\mathsf{br}(1) \oplus \mathsf{aux}{:}3.\mathsf{br}(1) \oplus \mathsf{aux}{:}2.\mathsf{br}(1) \oplus \mathsf{aux}{:}1.\mathsf{br}(0) \\
&= \mathsf{aux}{:}4.\mathsf{br}(1) \oplus \mathsf{aux}{:}3.\mathsf{br}(1) \oplus \mathsf{aux}{:}2.\mathsf{br}(0) \oplus \mathsf{aux}{:}1.\mathsf{br}(1)\ .
\end{aligned}$$

In this example, the behaviour of the instructions sequence under execution affects the Boolean registers from the Boolean register family such that it corresponds to decrement by one on the natural number represented by the combined content of the Boolean registers. The

equations show that, if the combined content of the Boolean registers represents 14, (a) the Boolean registers reduce the behaviour of the instruction sequence under execution to termination after four internal actions and (b) the behaviour of the instruction sequence under execution modifies the combined content of the Boolean registers to the binary representation of 13.

The following two propositions are about elimination results for closed $[\mathrm{PGA}/\mathrm{BTA}^{\infty}](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}$ terms.

Proposition 8. *For all closed* $[\mathrm{PGA}/\mathrm{BTA}^{\infty}](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}$ *terms* t *of sort* **T** *in which all subterms of sort* **IS** *are repetition-free, there exists a closed* $[\mathrm{PGA}/\mathrm{BTA}^{\infty}](\mathcal{A}_{\mathrm{br}})$ *term* t' *of sort* **T** *such that* $t = t'$ *is derivable from the axioms of* $[\mathrm{PGA}/\mathrm{BTA}^{\infty}](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}$.

Proof. It is easy to prove by structural induction that, for all closed repetition-free $[\mathrm{PGA}/\mathrm{BTA}^{\infty}](\mathcal{A}_{\mathrm{br}})$ terms s of sort **IS**, there exists a closed $[\mathrm{PGA}/\mathrm{BTA}^{\infty}](\mathcal{A}_{\mathrm{br}})$ term s' of sort **T** such that $|s| = s'$ is derivable from the axioms of $[\mathrm{PGA}/\mathrm{BTA}^{\infty}](\mathcal{A}_{\mathrm{br}})$. Therefore, it is sufficient to prove the proposition for all closed $[\mathrm{PGA}/\mathrm{BTA}^{\infty}](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}$ terms t of sort **T** in which no subterms of sort **IS** occur. This is proved similarly to part (1) of Theorem 3.1 from [2]. □

Proposition 9. *For all closed* $[\mathrm{PGA}/\mathrm{BTA}^{\infty}](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}$ *terms* t *of sort* **BRF** *in which all subterms of sort* **IS** *are repetition-free, there exists a closed* $[\mathrm{PGA}/\mathrm{BTA}^{\infty}](\mathcal{A}_{\mathrm{br}})$ *term* t' *of sort* **BRF** *such that* $t = t'$ *is derivable from the axioms of* $[\mathrm{PGA}/\mathrm{BTA}^{\infty}](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}$.

Proof. As in the proof of Proposition 8, it is sufficient to prove the proposition for all closed $[\mathrm{PGA}/\mathrm{BTA}^{\infty}](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}$ terms t of sort **BRF** in which no subterms of sort **IS** occur. This is proved similarly to part (2) of Theorem 3.1 from [2]. □

8 Computing Partial Functions from $\mathbb{B}^n$ to $\mathbb{B}^m$

In this section, we make precise in the setting of the algebraic theory $[\mathrm{PGA}/\mathrm{BTA}^{\infty}](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}$ what it means that a given instruction sequence computes a given partial function from $\mathbb{B}^n$ to $\mathbb{B}^m$ $(n, m \in \mathbb{N})$.

For each $n, m \in \mathbb{N}$, we define the following set:

$$\mathcal{F}_{\mathsf{br}}^{n,m} = \{\mathsf{in}{:}i \mid 1 \leq i \leq n\} \cup \{\mathsf{aux}{:}i \mid i \geq 1\} \cup \{\mathsf{out}{:}i \mid 1 \leq i \leq m\} \,.$$

We use the instantiation of $[\mathrm{PGA/BTA}^\infty](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}$ in which the set of foci is $\bigcup_{n,m\in\mathbb{N}} \mathcal{F}_{\mathsf{br}}^{n,m}$. We write $\mathcal{F}_{\mathsf{br}}$ for this set and we write $[[\mathrm{PGA/BTA}^\infty](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}](\mathcal{F}_{\mathsf{br}})$ for $[\mathrm{PGA/BTA}^\infty](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}$ with $\mathcal{F}$ instantiated by $\mathcal{F}_{\mathsf{br}}$.

Let $n, m \in \mathbb{N}$, let $F : \mathbb{B}^n \nrightarrow \mathbb{B}^m$,[7] and let t be a closed repetition-free $[[\mathrm{PGA/BTA}^\infty](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}](\mathcal{F}_{\mathsf{br}})$ term of sort **IS** in which only foci from $\mathcal{F}_{\mathsf{br}}^{n,m}$ occur. Then t *computes* F if there exists a $k \in \mathbb{N}$ such that:

- for all $b_1, \ldots, b_n, b'_1, \ldots, b'_m \in \mathbb{B}$ with $F(b_1, \ldots, b_n) = b'_1, \ldots, b'_m$:

$$(|t| \,/\, ((\textstyle\bigoplus_{i=1}^{n} \mathsf{in}{:}i.\mathsf{br}(b_i)) \oplus (\bigoplus_{i=1}^{k} \mathsf{aux}{:}i.\mathsf{br}(0)))) \bullet (\textstyle\bigoplus_{i=1}^{m} \mathsf{out}{:}i.\mathsf{br}(0)) = \textstyle\bigoplus_{i=1}^{m} \mathsf{out}{:}i.\mathsf{br}(b'_i) \;;$$

- for all $b_1, \ldots, b_n \in \mathbb{B}$ with $f(b_1, \ldots, b_n)$ undefined:

$$(|t| \,/\, ((\textstyle\bigoplus_{i=1}^{n} \mathsf{in}{:}i.\mathsf{br}(b_i)) \oplus (\bigoplus_{i=1}^{k} \mathsf{aux}{:}i.\mathsf{br}(0)))) \bullet (\textstyle\bigoplus_{i=1}^{m} \mathsf{out}{:}i.\mathsf{br}(0)) = \emptyset \,.$$

With this definition, we can establish whether an instruction sequence of the kind considered in $[[\mathrm{PGA/BTA}^\infty](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}](\mathcal{F}_{\mathsf{br}})$ computes a given partial function from $\mathbb{B}^n$ to $\mathbb{B}^m$ ($n, m \in \mathbb{N}$) by equational reasoning using the axioms of $[[\mathrm{PGA/BTA}^\infty](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}](\mathcal{F}_{\mathsf{br}})$.

The following proposition tells us that, for each partial function from $\mathbb{B}^n$ to $\mathbb{B}^m$, there exists an instruction sequence of the kind considered here that computes it.

Proposition 10. *For all $n, m \in \mathbb{N}$, for all $F : \mathbb{B}^n \nrightarrow \mathbb{B}^m$, there exists a closed repetition-free $[[\mathrm{PGA/BTA}^\infty](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}](\mathcal{F}_{\mathsf{br}})$ term t in which only basic instructions of the forms $f.0/0$, $f.1/1$, and $f.\mathrm{i}/\mathrm{i}$ with $f \in \mathcal{F}_{\mathsf{br}}^{n,m}$ occur such that t computes F.*

[7] We write $f : \mathbb{B}^n \nrightarrow \mathbb{B}^m$ to indicate that f is a partial function from $\mathbb{B}^n$ to $\mathbb{B}^m$.

Proof. As an immediate corollary of the proof of Theorem 5.6 in [2] we have the following: for all $n, m \in \mathbb{N}$, for all $F : \mathbb{B}^n \to \mathbb{B}^m$, there exists a closed repetition-free $[[\mathrm{PGA}/\mathrm{BTA}^\infty](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}](\mathcal{F}_{\mathsf{br}})$ term t in which only basic instructions of the forms $f.0/0$, $f.1/1$, and $f.\mathrm{i}/\mathrm{i}$ with $f \in \mathcal{F}_{\mathsf{br}}^{n,m}$ occur such that t computes F. It is easy to see from the same proof that this corollary generalizes from total functions to partial functions. □

The following proposition tells us that an instruction sequence in which not only basic instructions of the forms $f.0/0$, $f.1/1$, and $f.\mathrm{i}/\mathrm{i}$ occur can be transformed primitive instruction by primitive instruction to an at most linearly longer instruction sequence computing the same function in which only basic instructions of the forms $f.0/0$, $f.1/1$, and $f.\mathrm{i}/\mathrm{i}$ occur.

The *functional equivalence* relation $\sim_{\mathsf{f}}$ on the set of all closed repetition-free $[[\mathrm{PGA}/\mathrm{BTA}^\infty](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}](\mathcal{F}_{\mathsf{br}})$ terms of sort **IS** is defined by $t \sim_{\mathsf{f}} t'$ iff there exist $n, m \in \mathbb{N}$ such that:

- t and t' are terms in which only foci from $\mathcal{F}_{\mathsf{br}}^{n,m}$ occur;
- there exists a $F : \mathbb{B}^n \nrightarrow \mathbb{B}^m$ such that t computes F and t' computes F.

Proposition 11. *There exists a unary function ϕ on the set of all closed repetition-free $[[\mathrm{PGA}/\mathrm{BTA}^\infty](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}](\mathcal{F}_{\mathsf{br}})$ terms of sort* **IS** *such that:*

- *ϕ is the homomorphic extension of a function ϕ' from the set of all $[[\mathrm{PGA}/\mathrm{BTA}^\infty](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}](\mathcal{F}_{\mathsf{br}})$ constants of sort* **IS** *to the set of all closed repetition-free $[[\mathrm{PGA}/\mathrm{BTA}^\infty](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}](\mathcal{F}_{\mathsf{br}})$ terms of sort* **IS***;*
- *for all closed repetition-free $[[\mathrm{PGA}/\mathrm{BTA}^\infty](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}](\mathcal{F}_{\mathsf{br}})$ terms t of sort* **IS***:*
  - *$t \sim_{\mathsf{f}} \phi(t)$;*
  - *$\phi(t)$ is a term in which only basic instructions of the forms $f.0/0$, $f.1/1$, and $f.\mathrm{i}/\mathrm{i}$ occur;*

– *$\phi(t)$ is at most $3 \cdot p$ primitive instructions longer than t, where p is the number of occurrences of basic instructions in t that are not of the form $f.0/0$, $f.1/1$ or $f.\mathrm{i}/\mathrm{i}$.*

Proof. It follows immediately from part (3) of Proposition 3.1 in [2] that the definition of $\sim_{\mathsf{f}}$ given above is a reformulation of the instance of the definition of $\sim_{\mathsf{f}}$ given in [9], where the set $\mathcal{F}$ of foci is instantiated by $\mathcal{F}_{\mathsf{br}}$. This makes the current proposition a corollary of Proposition 2 and Theorem 3 in [9]. □

The view put forward in this section on what it means in the setting of $[\mathrm{PGA}/\mathrm{BTA}^{\infty}](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}$ that a given instruction sequence computes a given partial function from $\mathbb{B}^n$ to $\mathbb{B}^m$ $(n, m \in \mathbb{N})$ is the view taken in the work on complexity of computational problems, efficiency of algorithms, and algorithmic equivalence of programs presented in [3]–[8]. We remark that Boolean registers cannot only be used to compute partial functions from $\mathbb{B}^n$ to $\mathbb{B}^m$. For example, it is shown in [20] that jump instructions are not necessary if use can be made of Boolean registers.

9 Uses for the Theory

In this section, we give a survey of uses for $[\mathrm{PGA}/\mathrm{BTA}^{\infty}](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}$.

It is often said that a program is an instruction sequence and, if this characterization has any value, it must be the case that it is somehow easier to understand the concept of an instruction sequence than to understand the concept of a program. The first objective of the work on instruction sequences that started with [10], and of which an enumeration is available at [1], is to understand the concept of a program. The basis of all this work is the parameterized algebraic theory $\mathrm{PGA}/\mathrm{BTA}^{\infty}$ extended to deal with the interaction between instruction sequences under execution and components of their execution environment. The body of theory developed through this work is such that its use as a conceptual preparation for programming is practically feasible.

The notion of an instruction sequence appears in the work in question as a mathematical abstraction for which the rationale is based on

the objective mentioned above. In this capacity, instruction sequences constitute a primary field of investigation in programming comparable to propositions in logic and rational numbers in arithmetic. The structure of the mathematical abstraction at issue has been determined in advance with the hope of applying it in diverse circumstances where in each case the fit may be less than perfect.

Until now, the work in question has, among other things, yielded an approach to computational complexity, where program size is used as complexity measure, a contribution to the conceptual analysis of the notion of an algorithm, and new insights into such diverse issues as the halting problem, program parallelization for the purpose of explicit multi-threading and virus detection.

The work done in the setting of $[\mathrm{PGA}/\mathrm{BTA}^{\infty}](\mathcal{A}_{\mathrm{br}})/\mathrm{BRI}$, which is just an instantiation of the above-mentioned basis, includes:

- Work yielding an approach to computational complexity in which algorithmic problems are viewed as families of functions that consist of a function from $\mathbb{B}^n$ to $\mathbb{B}$ for each natural number n and the complexity of such problems is assessed in terms of the length of instruction sequences that compute the members of these families. Several kinds of non-uniform complexity classes have been introduced. One kind includes a counterpart of the well-known complexity class P/poly and another kind includes a counterpart of the well-known complexity class NP/poly (see [4]).

- Work contributing to the conceptual analysis of the notion of an algorithm. Two equivalence relations on instruction sequences have been defined, an algorithmic equivalence relation and a computational equivalence relation. The algorithmic equivalence relation captures to a reasonable degree the intuitive notion that two instruction sequences express the same algorithm. Any equivalence relation that captures the notion that two instruction sequences express the same algorithm to a higher degree must be finer than the computational equivalence relation (see [5]).

- Work showing that, in the case of computing the parity function on bit strings of length n, for each natural number n, shorter

instruction sequences are possible with the use of an auxiliary Boolean register than without the use of auxiliary Boolean registers. This result supports, in a setting where programs are instruction sequences acting on Boolean registers, a basic intuition behind the storage of auxiliary data, namely the intuition that this makes possible a reduction of the size of a program (see [6]).

- Work providing mathematically precise alternatives to the natural language and pseudo code descriptions of the long multiplication algorithm and the Karatsuba multiplication algorithm. One established result is that the instruction sequence expressing the latter algorithm is shorter than the instruction sequence expressing the former algorithm only if the length of the bit strings involved is greater than 256. Another result is that in a setting with backward jump instructions the long multiplication algorithm can be expressed by an instruction sequence that is shorter than both these instruction sequences if the length of the bit strings involved is greater than 2 (see [7]).

- Work showing that the problem of deciding whether an instruction sequence computes the function modeling the non-zeroness test on natural numbers less than 2^n with respect to their binary representation by bit strings of length n, for natural number n, can only be efficiently solved under the restriction that the length of the instruction sequence is close to the length of the shortest possible instruction sequences that compute this function (see [8]).

10 Concluding Remarks

We have presented the theory underlying a considerable part of the work done so far in a line of research in which issues relating to a wide variety of subjects from computer science are rigorously investigated thinking in terms of instruction sequences. The distinguishing feature of this presentation is that it is less involved than previous presenta-

tions. Sections 2, 3, and 4 concern the part of the presented theory that is relevant to all the work done so far in the line of research referred to.

The restriction to instructions that operate on Boolean registers is a classical restrictions in computer science. Other such classical restrictions are the restriction to instructions that operate on natural number registers in register machines and the restriction to instructions that operate on Turing tapes in Turing machines (see e.g. [21]). Adaptation of Sections 5, 6, and 7 to these restrictions is rather straightforward (cf. [12]).

Notice that we have fixed in Section 8, for each use of a Boolean register that must be distinguished to make precise what it means that a given instruction sequence computes a given partial function from $\mathbb{B}^n$ to $\mathbb{B}^m$ ($n, m \in \mathbb{N}$), the focus by which the Boolean register for that use is named. Because of this and the required generality, the possibility that the same Boolean register is used as both input register and output register is excluded. Exclusion of possibilities like this can be circumvented by abandoning the fixed assignment of foci to register uses and defining "t computes f" relative to an assignment of foci to register uses. This approach complicates matters, but seems indispensable to find conclusive answers to open questions like "what are the shortest instruction sequences that compute the function on bit strings of length n that models addition modulo 2^n on natural numbers less than 2^n, for $n \in \mathbb{N}_1$?".

The instruction sequences with instructions for Boolean registers considered in this paper constitute essentially a programming language in which all variables are Boolean variables. Such programming languages are actually used in toolkits for software model checking that make use of the abstract interpretation technique known as predicate abstraction (see e.g. [22], [23]).

Acknowledgements

We thank an anonymous referee for carefully reading a preliminary version of this paper, for pointing out some slips made in it, and for posing questions that have led to improvements of the presentation.

References

[1] C. A. Middelburg, "Instruction sequences as a theme in computer science," http://instructionsequence.wordpress.com/, 2015.

[2] J. A. Bergstra and C. A. Middelburg, *Instruction Sequences for Computer Science*, ser. Atlantis Studies in Computing. Amsterdam: Atlantis Press, 2012, vol. 2.

[3] J. A. Bergstra and C. A. Middelburg, "Instruction sequence expressions for the secure hash algorithm SHA-256," arXiv:1308.0219v7 [cs.PL], August 2013.

[4] J. A. Bergstra and C. A. Middelburg, "Instruction sequence based non-uniform complexity classes," *Scientific Annals of Computer Science*, vol. 24, no. 1, pp. 47–89, 2014.

[5] J. A. Bergstra and C. A. Middelburg, "On algorithmic equivalence of instruction sequences for computing bit string functions," *Fundamenta Informaticae*, vol. 138, no. 4, pp. 411–434, 2015.

[6] J. A. Bergstra and C. A. Middelburg, "Instruction sequence size complexity of parity," *Fundamenta Informaticae*, vol. 149, no. 3, pp. 297–309, 2016.

[7] J. A. Bergstra and C. A. Middelburg, "Instruction sequences expressing multiplication algorithms," To appear in Scientific Annals of Computer Science. Preprint: arXiv:1312.1529v4 [cs.PL], 2018.

[8] J. A. Bergstra and C. A. Middelburg, "On the complexity of the correctness problem for non-zeroness test instruction sequences," arXiv:1805.05845v1 [cs.LO], May 2018.

[9] J. A. Bergstra and C. A. Middelburg, "On instruction sets for Boolean registers in program algebra," *Scientific Annals of Computer Science*, vol. 26, no. 1, pp. 1–26, 2016.

[10] J. A. Bergstra and M. E. Loots, "Program algebra for sequential code," *Journal of Logic and Algebraic Programming*, vol. 51, no. 2, pp. 125–156, 2002.

[11] J. A. Bergstra and A. Ponse, "Combining programs and state machines," *Journal of Logic and Algebraic Programming*, vol. 51, no. 2, pp. 175–192, 2002.

[12] J. A. Bergstra and C. A. Middelburg, "Instruction sequence processing operators," *Acta Informatica*, vol. 49, no. 3, pp. 139–172, 2012.

[13] H. Ehrig and B. Mahr, *Fundamentals of Algebraic Specification I: Equations and Initial Semantics*, ser. EATCS Monographs. Berlin: Springer-Verlag, 1985, vol. 6.

[14] D. Sannella and A. Tarlecki, "Algebraic preliminaries," in *Algebraic Foundations of Systems Specification*, E. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner, Eds. Berlin: Springer-Verlag, 1999, pp. 13–30.

[15] D. Sannella and A. Tarlecki, *Foundations of Algebraic Specification and Formal Software Development*, ser. Monographs in Theoretical Computer Science, An EATCS Series. Berlin: Springer-Verlag, 2012.

[16] M. Wirsing, "Algebraic specification," in *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Amsterdam: Elsevier, 1990, vol. B, pp. 675–788.

[17] R. J. van Glabbeek and F. W. Vaandrager, "Modular specification of process algebras," *Theoretical Computer Science*, vol. 113, no. 2, pp. 293–348, 1993.

[18] J. A. Bergstra and C. A. Middelburg, "Axioms for behavioural congruence of single-pass instruction sequences," *Scientific Annals of Computer Science*, vol. 27, no. 2, pp. 111–135, 2017.

[19] G. B. Steven, S. M. Gray, and R. G. Adams, "HARP: A parallel pipelined RISC processor," *Microprocessors and Microsystems*, vol. 13, no. 9, pp. 579–587, 1989.

[20] J. A. Bergstra and C. A. Middelburg, "On the expressiveness of single-pass instruction sequences," *Theory of Computing Systems*, vol. 50, no. 2, pp. 313–328, 2012.

[21] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, 3rd ed. Reading, MA: Addison-Wesley, 2007.

[22] T. Ball and S. K. Rajamani, "Bebop: A symbolic model checker for Boolean programs," in *SPIN 2000*, ser. Lecture Notes in Computer Science, K. Havelund, J. Penix, and W. Visser, Eds., vol. 1885. Springer-Verlag, 2000, pp. 113–130.

[23] T. Ball and S. K. Rajamani, "The SLAM toolkit," in *CAV 2001*, ser. Lecture Notes in Computer Science, G. Berry, H. Comon, and A. Finkel, Eds., vol. 2102. Springer-Verlag, 2001, pp. 260–264.

Jan A. Bergstra, Cornelis A. Middelburg, Received August 16, 2018

Jan A. Bergstra
Informatics Institute, Faculty of Science, University of Amsterdam
Science Park 904, 1098 XH Amsterdam, the Netherlands
E–mail: J.A.Bergstra@uva.nl

Cornelis A. Middelburg
Informatics Institute, Faculty of Science, University of Amsterdam
Science Park 904, 1098 XH Amsterdam, the Netherlands
E–mail: C.A.Middelburg@uva.nl