On instruction sets for Boolean registers in program algebra

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

DOI
10.7561/SACS.2016.1.1

Publication date
2016

Document Version
Final published version

Published in
Scientific Annals of Computer Science

Scientific Annals of Computer Science vol. 26 (1), 2016, pp. 1–26
doi: 10.7561/SACS.2016.1.1

On Instruction Sets for Boolean Registers in Program Algebra

J.A. Bergstra[1], C.A. Middelburg[1]

Abstract

In previous work carried out in the setting of program algebra, including work in the area of instruction sequence size complexity, we chose instruction sets for Boolean registers that contain only instructions of a few of the possible kinds. In the current paper, we study instruction sequence size bounded functional completeness of all possible instruction sets for Boolean registers. We expect that the results of this study will turn out to be useful to adequately assess results of work that is concerned with lower bounds of instruction sequence size complexity.

Keywords: Boolean register, instruction set, size-bounded functional completeness, instruction sequence size, program algebra.

1 Introduction

In [5], we presented an approach to computational complexity in which algorithmic problems are viewed as Boolean function families that consist of one n-ary Boolean function for each natural number n and the complexity of such problems is assessed in terms of the length of finite single-pass instruction sequences acting on Boolean registers that compute the members of these families. The instruction sequences concerned contain only instructions to set and get the content of Boolean registers, forward jump instructions, and a termination instruction. Moreover, each Boolean register used serves as either input register, output register or auxiliary register.

[1] Informatics Institute, Faculty of Science, University of Amsterdam, Science Park 904, 1098 XH Amsterdam, the Netherlands, E-mail: {J.A.Bergstra,C.A.Middelburg}@uva.nl.

Auxiliary Boolean registers are not needed to compute Boolean functions. The question whether shorter instruction sequences are possible with the use of auxiliary Boolean registers was not answered in [5]. In [6], we showed that, in the case of the parity functions, shorter instruction sequences are possible with the use of an auxiliary Boolean register provided the instruction set is extended with instructions to complement the content of auxiliary Boolean registers. In the current paper, we consider all instructions for Boolean registers that are possible in the setting in which the work presented in [5, 6] has been carried out and investigate instruction sequence size bounded functional completeness of instruction sets for Boolean registers.

Intuitively, a given instruction set for Boolean registers is n-size-bounded functionally complete if the effects of each possible instruction for Boolean registers can be obtained by an instruction sequence whose length is at most n and that contains only instructions from the given instruction set for Boolean registers, forward jump instructions, and a termination instruction. A given instruction set for Boolean registers is functionally complete if it is n-size-bounded functionally complete for some n.

We have identified one of the 256 smallest instruction sets for Boolean registers that is 1-size-bounded functionally complete (Corollary 2), and we have found that there is a large subset of this 1-size-bounded functionally complete instruction set with the following properties: (i) each of its proper subsets that does not include the instructions to complement the content of Boolean registers, but includes the instructions to set and get the content of Boolean registers, is 4-size-bounded functionally complete and not 3-size-bounded functionally complete and (ii) each of its proper subsets that includes the instructions to complement the content of Boolean registers is 3-size-bounded functionally complete and not 2-size-bounded functionally complete (Corollary 3).

The use of a 1-size-bounded functionally complete instruction set, such as the one referred to in the previous paragraph, gives rise to the smallest instruction sequence sizes. However, the use of many instruction sets that are not 1-size-bounded functionally complete, e.g. the ones referred to under (i) above, gives rise to instruction sequence sizes that are at most 4 times larger.

The work presented in [6] triggered the work presented in this paper because of the choice to use an extension of the instruction set used in [5]. Since the results from the latter paper, with the exception of one auxiliary result, are concerned with upper bounds of instruction sequence size complexity, these results go through if instruction sequences may also contain instructions to complement the content of auxiliary Boolean registers. However, for the work presented in [6], the instruction set did matter in the sense that we were not able to prove the main result of the paper, which is concerned with a lower bound of instruction sequence size complexity, using the instruction set used in [5]. We consider the work presented in the current paper to be useful to adequately assess that result as it is. We expect that it will turn out to be also useful to adequately assess results of future work that is concerned with lower bounds of instruction sequence size complexity.

Like the work presented in [5, 6], the work presented in this paper is carried out in the setting of PGA (ProGram Algebra). PGA is an algebraic theory of single-pass instruction sequences that was taken as the basis of the approach to the semantics of programming languages introduced in [2]. As a continuation of the work presented in [2], (i) the notion of an instruction sequence was subjected to systematic and precise analysis and (ii) issues relating to diverse subjects in computer science and computer engineering were rigorously investigated in the setting of PGA. The subjects concerned include programming language expressiveness, computability, computational complexity, algorithm efficiency, algorithmic equivalence of programs, program verification, program compactness, micro-architecture, and probabilistic programming. For a comprehensive survey of a large part of this work, see [4]. An overview of all the work done to date in the setting of PGA and open questions originating from this work can be found on [11].

This paper is organized as follows. First, we present the preliminaries to the work presented in this paper (Sections 2 and 3) and introduce the possible instructions for Boolean registers (Section 4). Next, we define an equivalence relation on these instructions that identifies instructions that have the same effects (Section 5) and study instruction sequence size bounded functional completeness of instruction sets for Boolean registers (Section 6). Finally, we make some concluding remarks (Section 7).

Some familiarity with the basic notions related to algebraic theories and their models is assumed in this paper. The relevant notions are explained in handbook chapters and books on the foundations of algebraic specification, e.g. [8, 12, 13, 15].

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, where a better link up with commonly used terminology is expected, the elements of $\mathbb{B}$ are loosely called bits and the elements of $\mathbb{B}^n$ are loosely called bit strings of length n.

The preliminaries to the work presented in this paper (Sections 2 and 3) are almost the same as the preliminaries to the work presented in [7] and earlier papers. For this reason, there is some text overlap with those papers. Apart from the preliminaries, the material in this paper is new.

2 Program Algebra and Basic Thread Algebra

In this section, we give a survey of PGA (ProGram Algebra) and BTA (Basic Thread Algebra) and make precise in the setting of BTA which behaviours are produced by the instruction sequences considered in PGA under execution. The greater part of this section originates from [5]. A comprehensive introduction to PGA and BTA, including examples, can among other things be found in [4].

In PGA, it is assumed that there is a fixed but arbitrary set $\mathfrak{A}$ of *basic instructions*. The intuition is that the execution of a basic instruction may modify a state and produces a reply at its completion. The possible replies are 0 and 1. The actual reply is generally state-dependent. The set $\mathfrak{A}$ is the basis for the set of instructions that may occur in the instruction sequences considered in PGA. The elements of the latter set are called *primitive instructions*. There are five kinds of primitive instructions:

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

We write $\mathfrak{I}$ for the set of all primitive instructions.

On execution of 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.

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 $u \in \mathfrak{I}$, the *instruction* constant $u : \to \mathbf{IS}$;
- the binary *concatenation* operator $\_ \mathbin{;} \_ : \mathbf{IS} \times \mathbf{IS} \to \mathbf{IS}$;
- 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.

A closed PGA term is considered to denote a non-empty, finite or eventually periodic infinite sequence of primitive instructions.[2] The instruction sequence denoted by a closed term of the form $t \mathbin{;} t'$ is the instruction sequence denoted by t concatenated with the instruction sequence denoted by t'. The instruction sequence denoted by a closed term of the form t^{ω} is the instruction sequence denoted by t concatenated infinitely many times with itself.

Closed PGA terms are considered equal if they represent the same instruction sequence. The axioms for instruction sequence equivalence are given in Table 1. In this table, n stands for an arbitrary natural number

[2] An eventually periodic infinite sequence is an infinite sequence with only finitely many distinct suffixes.

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

from $\mathbb{N}_1$.[3] 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 $t^{n+1} = t ; t^n$.

A typical model of PGA is the model in which:

- the domain is the set of all finite and eventually periodic infinite sequences over the set $\mathfrak{I}$ of primitive instructions;
- the operation associated with $;$ is concatenation;
- the operation associated with $^\omega$ is the operation $^{\underline{\omega}}$ defined as follows:
  - if U is finite, then $U^{\underline{\omega}}$ is the unique infinite sequence U' such that U concatenated n times with itself is a proper prefix of U' for each $n \in \mathbb{N}$;
  - if U is infinite, then $U^{\underline{\omega}}$ is U.

It is immediately clear that this model has no proper subalgebra. Moreover, we know from [2, Section 3.2.2] that the axioms of PGA are complete with respect to satisfaction of equations between closed terms in this model. Hence, this model is an initial model of PGA (see e.g. [12]).

We confine ourselves to this model of PGA for the interpretation of PGA terms. In the sequel, we use the term *PGA instruction sequence* for the elements of the domain of this model. Below, we will use BTA to make precise which behaviours are produced by PGA instruction sequences under execution.

In BTA, it is assumed that a fixed but arbitrary set $\mathcal{A}$ of *basic actions* has been given. The objects considered in BTA are called threads. A thread represents a behaviour which consists of performing basic actions in a sequential fashion. Upon each basic action performed, a reply from an execution environment determines how the thread proceeds. The possible replies are the values 0 and 1.

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

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 every $a \in \mathcal{A}$, the binary *postconditional composition* operator $\_ \unlhd a \unrhd \_ : \mathbf{T} \times \mathbf{T} \to \mathbf{T}$.

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. We use infix notation for postconditional composition. We introduce *basic action prefixing* as an abbreviation: $a \circ t$, where t is a BTA term, abbreviates $t \unlhd a \unrhd t$. We identify expressions of the form $a \circ t$ with the BTA term they stand for.

The thread denoted by a closed term of the form $t \unlhd a \unrhd t'$ will first perform a, and then proceed as the thread denoted by t if the reply from the execution environment is 1 and proceed as the thread denoted by t' if the reply from the execution environment is 0. The thread denoted by S will do no more than terminate and the thread denoted by D will become inactive.

Closed BTA terms are considered equal if they are syntactically the same. Therefore, BTA has no axioms.

Each closed BTA term denotes 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. [3]). We are only interested in models of BTA in which sets of recursion equations have unique solutions, such as the projective limit model of BTA presented in [4].

We confine ourselves to this model of BTA, which has an initial model of BTA as a submodel, for the interpretation of BTA terms. In the sequel, we use the term *BTA thread* or simply *thread* for the elements of the domain of this model.

Regular threads, i.e. finite or infinite threads that can only be in a finite number of states, can be defined by means of a finite set of recursion equations. Provided that the set $\mathfrak{A}$ of basic instructions is identified with the set $\mathcal{A}$ of basic actions, the behaviours produced by PGA instruction sequences under execution are exactly the behaviours represented by regular threads and the behaviours produced by finite PGA instruction sequences are exactly the behaviours represented by finite threads.

Table 2: Axioms for the thread extraction operator

$\lvert a \rvert = a \circ \mathsf{D}$	$\lvert \#l \rvert = \mathsf{D}$
$\lvert a \,;\, X \rvert = a \circ \lvert X \rvert$	$\lvert \#0 \,;\, X \rvert = \mathsf{D}$
$\lvert +a \rvert = a \circ \mathsf{D}$	$\lvert \#1 \,;\, X \rvert = \lvert X \rvert$
$\lvert +a \,;\, X \rvert = \lvert X \rvert \trianglelefteq a \trianglerighteq \lvert \#2 \,;\, X \rvert$	$\lvert \#l+2 \,;\, u \rvert = \mathsf{D}$
$\lvert -a \rvert = a \circ \mathsf{D}$	$\lvert \#l+2 \,;\, u \,;\, X \rvert = \lvert \#l+1 \,;\, X \rvert$
$\lvert -a \,;\, X \rvert = \lvert \#2 \,;\, X \rvert \trianglelefteq a \trianglerighteq \lvert X \rvert$	$\lvert ! \rvert = \mathsf{S}$
	$\lvert ! \,;\, X \rvert = \mathsf{S}$

Henceforth, we will identify $\mathfrak{A}$ with $\mathcal{A}$. Intuitively, this means that we will not distinguish the basic action that takes place when a basic instruction is executed from that basic instruction.

We combine PGA with BTA, identifying $\mathfrak{A}$ with $\mathcal{A}$, and extend the combination with the *thread extraction* operator $\lvert \_ \rvert : \mathbf{IS} \to \mathbf{T}$, the axioms given in Table 2, and the rule that $\lvert X \rvert = \mathsf{D}$ if X has an infinite chain of forward jumps beginning at its first primitive instruction.[4] In Table 2, a stands for an arbitrary basic instruction from $\mathfrak{A}$, u stands for an arbitrary primitive instruction from $\mathfrak{I}$, and l stands for an arbitrary natural number from $\mathbb{N}$. For each closed PGA term t, $\lvert t \rvert$ denotes the behaviour produced by the instruction sequence denoted by t under execution.

3 Interaction of Threads with Services

Services are objects that represent the behaviours exhibited by components of execution environments of instruction sequences at a high level of abstraction. A service is able to process certain methods. For the purpose of the extension of BTA that will be presented in this section, it is sufficient to know the following about methods: (i) the processing of a method by a service may involve a change of the service and (ii) at completion of the processing of a method by a service, the service produces a reply value. The possible reply values are 0 and 1. Execution environments are considered to provide a family of uniquely-named services.

A thread may interact with the named services from the service family provided by an execution environment. That is, a thread may perform a

[4] This rule, which can be formalized using an auxiliary structural congruence predicate (see e.g. [4]), is unnecessary when considering only finite PGA instruction sequences.

basic action for the purpose of requesting a named service to process a method and to return a reply value at completion of the processing of the method. In this section, we give a survey of the extension of BTA with services, service families, a composition operator for service families, and operators that are concerned with this kind of interaction. This section originates from [3]. A comprehensive introduction to the presented extension of BTA, including examples, can among other things be found in [4].

First, we introduce an algebraic theory of service families called SFA (Service Family Algebra). In SFA, it is assumed that a fixed but arbitrary set $\mathcal{M}$ of *methods* has been given. Moreover, the following is assumed with respect to services:

- a signature $\Sigma_{\mathcal{S}}$ has been given that includes the following sorts:
  - the sort $\mathbf{S}$ of *services*;
  - the sort $\mathbf{R}$ of *replies*;

  and the following constants and operators:
  - the *empty service* constant $\delta : \to \mathbf{S}$;
  - the *reply* constants $0, 1, * : \to \mathbf{R}$;
  - for each $m \in \mathcal{M}$, the *derived service* operator $\frac{\partial}{\partial m} : \mathbf{S} \to \mathbf{S}$;
  - for each $m \in \mathcal{M}$, the *service reply* operator $\varrho_m : \mathbf{S} \to \mathbf{R}$;
- a $\Sigma_{\mathcal{S}}$-algebra $\boldsymbol{\mathcal{S}}$ that has no proper subalgebra has been given in which the following holds:
  - $0 \neq 1$, $1 \neq *$, $* \neq 0$;
  - for each $m \in \mathcal{M}$, $\frac{\partial}{\partial m}(z) = \delta \Leftrightarrow \varrho_m(z) = *$.

The intuition concerning $\frac{\partial}{\partial m}$ and ϱ_m is that on a request to service s to process method m:

- if $\varrho_m(s) \neq *$, s processes m, produces the reply $\varrho_m(s)$, and then proceeds as $\frac{\partial}{\partial m}(s)$;
- if $\varrho_m(s) = *$, s is not able to process method m and proceeds as δ.

The empty service δ itself is unable to process any method.

It is also assumed that a fixed but arbitrary set $\mathcal{F}$ of *foci* has been given. Foci play the role of names of services in a service family.

SFA has the sorts, constants and operators from $\Sigma_{\mathcal{S}}$ and in addition the sort $\mathbf{SF}$ of *service families* and the following constant and operators:

Table 3: Axioms of SFA

$u \oplus \emptyset = u$	SFC1	$\partial_F(\emptyset) = \emptyset$	SFE1
$u \oplus v = v \oplus u$	SFC2	$\partial_F(f.z) = \emptyset \quad \textsf{if } f \in F$	SFE2
$(u \oplus v) \oplus w = u \oplus (v \oplus w)$	SFC3	$\partial_F(f.z) = f.z \textsf{ if } f \notin F$	SFE3
$f.z \oplus f.z' = f.\delta$	SFC4	$\partial_F(u \oplus v) = \partial_F(u) \oplus \partial_F(v)$	SFE4

- the *empty service family* constant $\emptyset : \to \mathbf{SF}$;
- for each $f \in \mathcal{F}$, the unary *singleton service family* operator $f.\_ : \mathbf{S} \to \mathbf{SF}$;
- the binary *service family composition* operator $\_ \oplus \_ : \mathbf{SF} \times \mathbf{SF} \to \mathbf{SF}$;
- for each $F \subseteq \mathcal{F}$, the unary *encapsulation* operator $\partial_F : \mathbf{SF} \to \mathbf{SF}$.

We assume that there are infinitely many variables of sort $\mathbf{S}$, including z, and infinitely many variables of sort $\mathbf{SF}$, including u, v, w. Terms are built as usual in the many-sorted case (see e.g. [13]). We use prefix notation for the singleton service family operators and infix notation for the service family composition operator. We write $\bigoplus_{i=1}^{n} t_i$, where $t_1, \ldots, t_n$ are terms of sort $\mathbf{SF}$, for the term $t_1 \oplus \ldots \oplus t_n$.

The service family denoted by $\emptyset$ is the empty service family. The service family denoted by a closed term of the form $f.t$ consists of one named service only, the service concerned is the service denoted by t, and it is named f. The service family denoted by a closed term of the form $t \oplus t'$ consists of all named services that belong to either the service family denoted by t or the service family denoted by t'. In the case where a named service from the service family denoted by t and a named service from the service family denoted by t' have the same name, they collapse to an empty service with the name concerned. The service family denoted by a closed term of the form $\partial_F(t)$ consists of all named services with a name not in F that belong to the service family denoted by t.

The axioms of SFA are given in Table 3. In this table, f stands for an arbitrary focus from $\mathcal{F}$ and F stands for an arbitrary subset of $\mathcal{F}$. These axioms simply formalize the informal explanation given above.

For the set $\mathcal{A}$ of basic actions, we now take $\{f.m \mid f \in \mathcal{F}, m \in \mathcal{M}\}$. Performing a basic action $f.m$ is taken as making a request to the service named f to process method m.

Table 4: Axioms for the abstracting use operator

$\mathsf{S} \mathbin{/\!\!/} u = \mathsf{S}$		AU1
$\mathsf{D} \mathbin{/\!\!/} u = \mathsf{D}$		AU2
$(x \unlhd f.m \unrhd y) \mathbin{/\!\!/} \partial_{\{f\}}(u) = (x \mathbin{/\!\!/} \partial_{\{f\}}(u)) \unlhd f.m \unrhd (y \mathbin{/\!\!/} \partial_{\{f\}}(u))$		AU3
$(x \unlhd f.m \unrhd y) \mathbin{/\!\!/} (f.t \oplus \partial_{\{f\}}(u)) = x \mathbin{/\!\!/} (f.\frac{\partial}{\partial m}t \oplus \partial_{\{f\}}(u))$	if $\varrho_m(t) = 1$	AU4
$(x \unlhd f.m \unrhd y) \mathbin{/\!\!/} (f.t \oplus \partial_{\{f\}}(u)) = y \mathbin{/\!\!/} (f.\frac{\partial}{\partial m}t \oplus \partial_{\{f\}}(u))$	if $\varrho_m(t) = 0$	AU5
$(x \unlhd f.m \unrhd y) \mathbin{/\!\!/} (f.t \oplus \partial_{\{f\}}(u)) = \mathsf{D}$	if $\varrho_m(t) = *$	AU6

Table 5: Axioms for the apply operator

$\mathsf{S} \bullet u = u$		A1
$\mathsf{D} \bullet u = \emptyset$		A2
$(x \unlhd f.m \unrhd y) \bullet \partial_{\{f\}}(u) = \emptyset$		A3
$(x \unlhd f.m \unrhd y) \bullet (f.t \oplus \partial_{\{f\}}(u)) = x \bullet (f.\frac{\partial}{\partial m}t \oplus \partial_{\{f\}}(u))$	if $\varrho_m(t) = 1$	A4
$(x \unlhd f.m \unrhd y) \bullet (f.t \oplus \partial_{\{f\}}(u)) = y \bullet (f.\frac{\partial}{\partial m}t \oplus \partial_{\{f\}}(u))$	if $\varrho_m(t) = 0$	A5
$(x \unlhd f.m \unrhd y) \bullet (f.t \oplus \partial_{\{f\}}(u)) = \emptyset$	if $\varrho_m(t) = *$	A6

We combine BTA with SFA and extend the combination with the following operators:

- the binary *abstracting use* operator $\_ \mathbin{/\!\!/} \_ : \mathbf{T} \times \mathbf{SF} \to \mathbf{T}$;
- the binary *apply* operator $\_ \bullet \_ : \mathbf{T} \times \mathbf{SF} \to \mathbf{SF}$;

and the axioms given in Tables 4 and 5. In these tables, f stands for an arbitrary focus from $\mathcal{F}$, m stands for an arbitrary method from $\mathcal{M}$, and t stands for an arbitrary term of sort $\mathbf{S}$. The axioms formalize the informal explanation given below and in addition stipulate what is the result of abstracting use and apply if inappropriate foci or methods are involved. We use infix notation for the abstracting use and apply operators.

The thread denoted by a closed term of the form $t \mathbin{/\!\!/} t'$ and the service family denoted by a closed term of the form $t \bullet t'$ are the thread and service family, respectively, that result from processing the method of each basic action performed by the thread denoted by t by the service in the service family denoted by t' with the focus of the basic action as its name if such a service exists. When the method of a basic action performed by a thread is processed by a service, the service changes in accordance with the method concerned and the thread reduces to one of the two threads that it can possibly proceed with dependent on the reply value produced by the service.

The projective limit model of the extension of the combination of BTA and SFA with the abstracting use operator, the apply operator, and the axioms for these operators is a reduct of the projective limit model presented in [4, Section 3.1.9]. The reduct of this model to the constants and operators of BTA is the projective limit model of BTA.

4 Instructions for Boolean Registers

The primitive instructions that concern us in the remainder of this paper are primitive instructions for Boolean registers. We introduce in this section the possible primitive instructions for Boolean registers.

It is assumed that, for each $p, q : \mathbb{B} \to \mathbb{B}$, $p/q \in \mathcal{M}$. These methods can be explained as follows:

> when p/q is processed by a Boolean register service whose register content is b, the reply is $p(b)$ and the register content becomes $q(b)$.

We write $\mathcal{M}_{\mathsf{br}}$ for the set $\{p/q \mid p, q : \mathbb{B} \to \mathbb{B}\}$. Every method that a Boolean register service could possibly process is a method from $\mathcal{M}_{\mathsf{br}}$.

For $\Sigma_{\mathcal{S}}$, we take the signature that consists of the sorts, constants and operators that are mentioned in the assumptions with respect to services made in Section 3 and a constant BR_b^M for each $M \subseteq \mathcal{M}_{\mathsf{br}}$ and $b \in \mathbb{B}$. Informally, BR_b^M denotes the Boolean register service with register content b that is able to process precisely all methods from M.

For $\boldsymbol{\mathcal{S}}$, we take the $\Sigma_{\mathcal{S}}$-algebra that has no proper subalgebra and that satisfies the conditions that are mentioned in the assumptions with respect to services made in Section 3 and the following conditions for each $M \subseteq \mathcal{M}_{\mathsf{br}}$ and $b \in \mathbb{B}$:

$$\begin{aligned} &\tfrac{\partial}{\partial p/q}(BR_b^M) = BR_{q(b)}^M \text{ if } p/q \in M\ , && \tfrac{\partial}{\partial m}(BR_b^M) = \delta \text{ if } m \notin M\ , \\ &\varrho_{p/q}(BR_b^M) = p(b) \quad\ \text{if } p/q \in M\ , && \varrho_m(BR_b^M) = * \text{ if } m \notin M\ . \end{aligned}$$

$\mathbb{B} \to \mathbb{B}$, 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 [5], we actually used the methods $0/0$, $1/1$, 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 methods, the method c/c, but denoted it by com.

We define, for each $M \subseteq \mathcal{M}_{\mathsf{br}}$, the following sets:

$$\mathcal{A}_{\mathsf{br}}(M) = \{f.m \mid f \in \mathcal{F} \land m \in M\} ,$$

$$\mathcal{I}_{\mathsf{br}}(M) = \mathcal{A}_{\mathsf{br}}(M) \cup \{+a \mid a \in \mathcal{A}_{\mathsf{br}}(M)\} \cup \{-a \mid a \in \mathcal{A}_{\mathsf{br}}(M)\} .$$

$\mathcal{A}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ consists of 16 basic actions per focus and $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ consists of 48 primitive instructions per focus.

For Boolean registers that serve as input register, we used in [5, 6] only primitive instructions from $\mathcal{I}_{\mathsf{br}}(\{\mathrm{i}/\mathrm{i}\})$. For Boolean registers that serve as output register, we used in [5, 6] only primitive instructions from $\mathcal{I}_{\mathsf{br}}(\{0/0, 1/1\})$. For Boolean registers that serve as auxiliary register, we used in [5] only primitive instructions from $\mathcal{I}_{\mathsf{br}}(\{0/0, 1/1, \mathrm{i}/\mathrm{i}\})$ and in [6] only primitive instructions from $\mathcal{I}_{\mathsf{br}}(\{0/0, 1/1, \mathrm{i}/\mathrm{i}, \mathrm{c}/\mathrm{c}\})$. However, in the case of auxiliary registers, other possible instruction sets are eligible. In Section 6, we study instruction sequence size-bounded functional completeness of instruction sets for Boolean registers. We expect that the results of that study will turn out to be useful to adequately assess results of work that is concerned with lower bounds of instruction sequence size complexity in cases where auxiliary Boolean registers may be used.

We write $\mathcal{IS}_{\mathsf{br}}(M)$, where $M \subseteq \mathcal{M}_{\mathsf{br}}$, for the set of all finite PGA instruction sequences in the case where $\mathcal{A}_{\mathsf{br}}(M)$ is taken for the set $\mathfrak{A}$ of basic instructions.

5 Equivalence of Instructions for Boolean Registers

There exists a model of the extension of the combination of PGA, BTA, and SFA with the thread extraction operator, the abstracting use operator, the apply operator, and the axioms for these operators such that the initial model of PGA is its reduct to the signature of PGA and the projective limit model of the extension of the combination of BTA and SFA with the abstracting use operator, the apply operator, and the axioms for these operators is its reduct to the signature of this theory. This follows from the disjointness of the signatures concerned by the amalgamation result about expansions presented as Theorem 6.1.1 in [9] (adapted to the many-sorted case).

Henceforth, we work in the model just mentioned, and denote the interpretations of constants and operators in it by the constants and operators themselves. However, we could work in any model for which the axioms are complete with respect to satisfaction of equations between closed terms.

On execution of an instruction sequence, different primitive instructions from $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ do not always have different effects. We define an equivalence on $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ that identifies primitive instructions if they have the same effects.

Let $u, v \in \mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$. Then u and v are *effectually equivalent*, written $u \sim_{\mathsf{e}} v$, if there exists an $f \in \mathcal{F}$ such that, for each $b \in \mathbb{B}$ and $n \in \{1, 2\}$:

$$|u\,;!^n| \mathbin{/\!/} f.BR_b^{\mathcal{M}_{\mathsf{br}}} = |v\,;!^n| \mathbin{/\!/} f.BR_b^{\mathcal{M}_{\mathsf{br}}}\,,$$
$$|u\,;!^n| \bullet f.BR_b^{\mathcal{M}_{\mathsf{br}}} = |v\,;!^n| \bullet f.BR_b^{\mathcal{M}_{\mathsf{br}}}\,.$$

Let $u, v \in \mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ be such that $u \neq v$, and let $f \in \mathcal{F}$ be such that, for some $m \in \mathcal{M}_{\mathsf{br}}$, $v \equiv f.m$ or $v \equiv +f.m$ or $v \equiv -f.m$. Then $u \sim_{\mathsf{e}} v$ only if $|u;!^n| \mathbin{/\!/} f.BR_b^{\mathcal{M}_{\mathsf{br}}} = |v;!^n| \mathbin{/\!/} f.BR_b^{\mathcal{M}_{\mathsf{br}}}$ and $|u;!^n| \bullet f.BR_b^{\mathcal{M}_{\mathsf{br}}} = |v;!^n| \bullet f.BR_b^{\mathcal{M}_{\mathsf{br}}}$ for each $b \in \mathbb{B}$ and $n \in \{1, 2\}$. From this and the definition of $\sim_{\mathsf{e}}$, it follows immediately that $\sim_{\mathsf{e}}$ is transitive. Moreover, it follows immediately from the definition of $\sim_{\mathsf{e}}$ that $\sim_{\mathsf{e}}$ is reflexive and symmetric. Hence, $\sim_{\mathsf{e}}$ is an equivalence relation indeed.

Replacement of primitive instructions in an instruction sequence by effectually equivalent ones does not change the functionality of the instruction sequence on execution.

Let $X, Y \in \mathcal{IS}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$. Then X and Y are *functionally equivalent*, written $X \sim_{\mathsf{f}} Y$, if, for some $n \in \mathbb{N}_1$, there exist $f_1, \ldots, f_n \in \mathcal{F}$ such that, for each $b_1, \ldots, b_n \in \mathbb{B}$:

$$|X| \mathbin{/\!/} \textstyle\bigoplus_{i=1}^{n} f_i.BR_{b_i}^{\mathcal{M}_{\mathsf{br}}} = \mathsf{S} \text{ or } |X| \mathbin{/\!/} \bigoplus_{i=1}^{n} f_i.BR_{b_i}^{\mathcal{M}_{\mathsf{br}}} = \mathsf{D},$$
$$|X| \mathbin{/\!/} \textstyle\bigoplus_{i=1}^{n} f_i.BR_{b_i}^{\mathcal{M}_{\mathsf{br}}} = |Y| \mathbin{/\!/} \bigoplus_{i=1}^{n} f_i.BR_{b_i}^{\mathcal{M}_{\mathsf{br}}},$$
$$|X| \bullet \textstyle\bigoplus_{i=1}^{n} f_i.BR_{b_i}^{\mathcal{M}_{\mathsf{br}}} = |Y| \bullet \bigoplus_{i=1}^{n} f_i.BR_{b_i}^{\mathcal{M}_{\mathsf{br}}}.$$

The proof that $\sim_{\mathsf{f}}$ is an equivalence relation goes along similar lines as the proof that $\sim_{\mathsf{e}}$ is an equivalence relation. Here, $X \sim_{\mathsf{f}} Y$ only if the

Table 6: Axioms for effectual equivalence

$+f.0/p \sim_{\mathsf{e}} -f.1/p$	$+f.1/p \sim_{\mathsf{e}} f.q/p$	$u \sim_{\mathsf{e}} u$
$+f.1/p \sim_{\mathsf{e}} -f.0/p$		$u \sim_{\mathsf{e}} v \;\Rightarrow\; v \sim_{\mathsf{e}} u$
$+f.\mathrm{i}/p \sim_{\mathsf{e}} -f.\mathrm{c}/p$		$u \sim_{\mathsf{e}} v \land v \sim_{\mathsf{e}} w \;\Rightarrow\; u \sim_{\mathsf{e}} w$
$+f.\mathrm{c}/p \sim_{\mathsf{e}} -f.\mathrm{i}/p$		

equations from the definition hold in the case where we take the foci of primitive instructions from $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ that occur in Y for $f_1, \ldots, f_n$.

Proposition 1 *Let $u, v \in \mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$, and let $X, Y \in \mathcal{IS}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ be such that Y is X with every occurrence of u replaced by v. Then $u \sim_{\mathsf{e}} v$ implies $X \sim_{\mathsf{f}} Y$.*

Proof: It is easily proved by induction on the length of X that $u \sim_{\mathsf{e}} v$ implies, for each $l, n \in \mathbb{N}$, $\#l \,;\, X \,;\, !^n \sim_{\mathsf{f}} \#l \,;\, Y \,;\, !^n$.[5] From this, it follows immediately that $u \sim_{\mathsf{e}} v$ implies $X \sim_{\mathsf{f}} Y$. □

Axioms for effectual equivalence are given in Table 6. In this table, f stands for an arbitrary focus from $\mathcal{F}$, p and q stand for arbitrary functions from $\mathbb{B} \to \mathbb{B}$, and u, v, and w stand for arbitrary primitive instructions from $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$. Moreover, we use $\sim_{\mathsf{e}}$ in this table as a predicate symbol (and not as the symbol that denotes the effectual equivalence relation on $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ defined above).

Theorem 1 *The axioms in Table 6 are sound and complete for the effectual equivalence relation on $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ defined above.*

Proof: The soundness of the axioms follows immediately from the definition of effectual equivalence, using the conditions on $\boldsymbol{\mathcal{S}}$ laid down in Section 4.

The following conclusions can be drawn from the definition of effectual equivalence:

$$
\begin{array}{ll}
\text{(a)} & +f.p/q \sim_{\mathsf{e}} f.p'/q' \;\Rightarrow\; p = 1 \land q = q'\,; \\
\text{(b)} & -f.p/q \sim_{\mathsf{e}} f.p'/q' \;\Rightarrow\; p = 0 \land q = q'\,; \\
\text{(c)} & +f.p/q \sim_{\mathsf{e}} -f.p'/q' \;\Rightarrow\; p = C(p') \land q = q'\,, \\
 & \text{where } C(0) = 1,\ C(1) = 0,\ C(\mathrm{i}) = \mathrm{c},\ C(\mathrm{c}) = \mathrm{i}.
\end{array}
$$

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

The completeness of the axioms follows easily by case distinction between the different forms that a formula $u \sim_{\mathsf{e}} v$ can take, making use of (a), (b), and (c). □

The equivalence classes of $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ with respect to $\sim_{\mathsf{e}}$ are the following for each $f \in \mathcal{F}$:

$$\begin{aligned}
&\{\underline{+f.0/0}, -f.1/0\}\;, \\
&\{+f.0/1, \underline{-f.1/1}\}\;, \\
&\{+f.0/\mathrm{i}, \underline{-f.1/\mathrm{i}}\}\;, \\
&\{+f.0/\mathrm{c}, \underline{-f.1/\mathrm{c}}\}\;, \\
&\{+f.1/0, -f.0/0, \underline{f.0/0}, f.1/0, f.\mathrm{i}/0, f.\mathrm{c}/0\}\;, \\
&\{+f.1/1, -f.0/1, f.0/1, \underline{f.1/1}, f.\mathrm{i}/1, f.\mathrm{c}/1\}\;, \\
&\{+f.1/\mathrm{i}, -f.0/\mathrm{i}, f.0/\mathrm{i}, f.1/\mathrm{i}, \underline{f.\mathrm{i}/\mathrm{i}}, f.\mathrm{c}/\mathrm{i}\}\;, \\
&\{+f.1/\mathrm{c}, -f.0/\mathrm{c}, f.0/\mathrm{c}, f.1/\mathrm{c}, f.\mathrm{i}/\mathrm{c}, \underline{f.\mathrm{c}/\mathrm{c}}\}\;, \\
&\{\underline{+f.\mathrm{i}/0}, -f.\mathrm{c}/0\}\;, \\
&\{\underline{+f.\mathrm{i}/1}, -f.\mathrm{c}/1\}\;, \\
&\{\underline{+f.\mathrm{i}/\mathrm{i}}, -f.\mathrm{c}/\mathrm{i}\}\;, \\
&\{+f.\mathrm{i}/\mathrm{c}, \underline{-f.\mathrm{c}/\mathrm{c}}\}\;, \\
&\{+f.\mathrm{c}/0, \underline{-f.\mathrm{i}/0}\}\;, \\
&\{+f.\mathrm{c}/1, \underline{-f.\mathrm{i}/1}\}\;, \\
&\{+f.\mathrm{c}/\mathrm{i}, \underline{-f.\mathrm{i}/\mathrm{i}}\}\;, \\
&\{\underline{+f.\mathrm{c}/\mathrm{c}}, -f.\mathrm{i}/\mathrm{c}\}\;.
\end{aligned}$$

We have underlined one representative of each equivalence class in order to refer to them easily in the proof of the following theorem.

Theorem 2

(1) *The set* $\{0/0, 1/1, \mathrm{i}/\mathrm{i}, \mathrm{c}/\mathrm{c}, \mathrm{i}/0, \mathrm{i}/1, 1/\mathrm{i}, 1/\mathrm{c}\}$ *is a minimal* $M \subseteq \mathcal{M}_{\mathsf{br}}$ *such that* $\mathcal{I}_{\mathsf{br}}(M)$ *contains at least one representative from each of the equivalence classes of* $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ *with respect to* $\sim_{\mathsf{e}}$.

(2) *Each minimal* $M \subseteq \mathcal{M}_{\mathsf{br}}$ *such that* $\mathcal{I}_{\mathsf{br}}(M)$ *contains at least one representative from each of the equivalence classes of* $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ *with respect to* $\sim_{\mathsf{e}}$ *consists of* 8 *methods.*

Proof: By uniformity, it is sufficient to look at the equivalence classes of $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ for an arbitrary focus from $\mathcal{F}$.

(1) Let $M' = \{0/0, 1/1, \mathrm{i}/\mathrm{i}, \mathrm{c}/\mathrm{c}, \mathrm{i}/0, \mathrm{i}/1, 1/\mathrm{i}, 1/\mathrm{c}\}$. Then the representatives of the different equivalence classes of $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ that are underlined above belong to the set $\mathcal{I}_{\mathsf{br}}(M')$. Moreover, each method from M' occurs in a primitive instruction from $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ that belongs to an equivalence class of $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ that contains only one other primitive instruction, but the method that occurs in this other primitive instruction is not from M'. Hence M' is minimal.

(2) First, we consider the first and last four equivalence classes above. Each of them consists of two primitive instructions. Each method that occurs in the primitive instructions from one of them does not occur in the primitive instructions from another of them. Consequently, exactly eight methods are needed for representatives from these equivalence classes. Next, we consider the remaining eight equivalence classes. For each of them, the methods that occur in the primitive instructions from it include the methods that occur in the primitive instructions from one of the equivalence classes that we considered first. Consequently, no additional methods are needed for representatives from the remaining equivalence classes. Hence, exactly eight methods are needed for representatives from all equivalence classes.

□

Theorem 2 tells us that each primitive instruction from $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ has the same effects as one with a method from $\{0/0, 1/1, \mathrm{i}/\mathrm{i}, \mathrm{c}/\mathrm{c}, \mathrm{i}/0, \mathrm{i}/1, 1/\mathrm{i}, 1/\mathrm{c}\}$ and that there does not exist a smaller set with this property. The methods that we used in [5, 6] are included in this set.

We have the following corollary of the proof of part (2) of Theorem 2.

Corollary 1 *There exist* 256 *minimal* $M \subseteq \mathcal{M}_{\mathsf{br}}$ *such that* $\mathcal{I}_{\mathsf{br}}(M)$ *contains at least one representative from each of the equivalence classes of* $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ *with respect to* $\sim_{\mathsf{e}}$.

6 Bounded Functional Completeness of Instruction Sets

Not all methods from the minimal set mentioned in Theorem 2 are needed to obtain the effects of each primitive instruction from $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ in the case where instruction sequences instead of instructions are used to obtain the effects. In this section, we look at the case where instruction sequences are used. We begin by defining the notion of k-size-bounded functional completeness.

Let $M \subseteq \mathcal{M}_{\mathsf{br}}$ and $k \in \mathbb{N}_1$. Then the instruction set $\mathcal{I}_{\mathsf{br}}(M)$ is *k-size-bounded functionally complete* if there exists a function $\psi : \mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}}) \to \mathcal{IS}_{\mathsf{br}}(M)$ such that, for each $u \in \mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$, $\mathrm{len}(\psi(u)) \leq k$ and there exists an $f \in \mathcal{F}$ such that, for each $b \in \mathbb{B}$ and $n \in \mathbb{N}$:

$$\begin{aligned} |u\,;{!}^n| \mathbin{/\!\!/} f.BR_b^{\mathcal{M}_{\mathsf{br}}} &= |\psi(u)\,;{!}^n| \mathbin{/\!\!/} f.BR_b^{\mathcal{M}_{\mathsf{br}}}\,,\\ |u\,;{!}^n| \bullet f.BR_b^{\mathcal{M}_{\mathsf{br}}} &= |\psi(u)\,;{!}^n| \bullet f.BR_b^{\mathcal{M}_{\mathsf{br}}}\,. \end{aligned}$$

$\mathcal{I}_{\mathsf{br}}(M)$ is called *strictly k-size-bounded functionally complete* if $\mathcal{I}_{\mathsf{br}}(M)$ is k-size-bounded functionally complete and there does not exist a $k' < k$ such that $\mathcal{I}_{\mathsf{br}}(M)$ is k'-size-bounded functionally complete.

The following proposition illustrates the relevance of the notion of k-size-bounded functionally completeness.

Proposition 2 *Let $M \subseteq \mathcal{M}_{\mathsf{br}}$ and $k \in \mathbb{N}_1$. Let $\psi : \mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}}) \to \mathcal{IS}_{\mathsf{br}}(M)$ be as in the definition of k-size-bounded functional completeness given above. Let $\psi' : \mathcal{IS}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}}) \to \mathcal{IS}_{\mathsf{br}}(M)$ be such that $\psi'(u_1\,;\ldots;u_n) = u'_1\,;\ldots;u'_n$, where*

$$\begin{array}{ll} u'_i \equiv {!} & \textit{if } u_i \equiv {!}\,;\\ u'_i \equiv \#l' & \\ \quad \textit{with } l' = l + \sum_{j \in \{i,\ldots,i+l-1\}\ \mathrm{s.t.}\ u_j \in \mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})} (\mathrm{len}(\psi(u_j)) - 1) & \textit{if } u_i \equiv \#l\,;\\ u'_i \equiv \psi(u_i) & \textit{otherwise.} \end{array}$$

Assume that ψ restricted to $\mathcal{IS}_{\mathsf{br}}(M)$ is the identity function on $\mathcal{IS}_{\mathsf{br}}(M)$. Then, for each $X \in \mathcal{IS}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$, $\psi'(X) \sim_{\mathsf{f}} X$ and $\mathrm{len}(\psi'(X)) \leq \mathrm{len}(X) + (k-1)\cdot p$, where p is the number of occurrences of primitive instructions from $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}}) \setminus \mathcal{I}_{\mathsf{br}}(M)$ in X.

Proof: It is easily proved by induction on the length of X that, for each $l, n \in \mathbb{N}$, $\#l\,;\psi'(X)\,;{!}^n \sim_{\mathsf{f}} \#l\,;X\,;{!}^n$. From this, it follows immediately that $\psi'(X) \sim_{\mathsf{f}} X$.

Suppose that $X = u_1\,;\ldots;u_n$. Let p be the number of occurrences of primitive instructions from $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}}) \setminus \mathcal{I}_{\mathsf{br}}(M)$ in X. Then

$$\begin{aligned}\textstyle\sum_{i \in \{1,\ldots,n\}\ \text{s.t.}\ u_i \notin \mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}}) \setminus \mathcal{I}_{\mathsf{br}}(M)} \mathrm{len}(u_i) &= \mathrm{len}(X) - p\ ; \\ \textstyle\sum_{i \in \{1,\ldots,n\}\ \text{s.t.}\ u_i \in \mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}}) \setminus \mathcal{I}_{\mathsf{br}}(M)} \mathrm{len}(\psi(u_i)) &\leq k \cdot p\ .\end{aligned}$$

Hence, $\mathrm{len}(\psi'(X)) \leq \mathrm{len}(X) - p + k \cdot p = \mathrm{len}(X) + (k-1) \cdot p$. □

We have the following corollary of part (1) of Theorem 2 and the definition of k-size-bounded functional completeness.

Corollary 2 $\mathcal{I}_{\mathsf{br}}(\{0/0, 1/1, \mathrm{i}/\mathrm{i}, \mathrm{c}/\mathrm{c}, \mathrm{i}/0, \mathrm{i}/1, 1/\mathrm{i}, 1/\mathrm{c}\})$ *is* 1*-size-bounded functionally complete.*

The following theorem concerns the k-size-bounded functional completeness of a few subsets of this 1-size-bounded functionally complete instruction set, including the ones that we used in [5, 6].

Theorem 3

(1) $\mathcal{I}_{\mathsf{br}}(\{0/0, 1/1, \mathrm{i}/\mathrm{i}, \mathrm{c}/\mathrm{c}, \mathrm{i}/0, \mathrm{i}/1\})$ *is strictly* 2*-size-bounded funct. compl.*

(2) $\mathcal{I}_{\mathsf{br}}(\{0/0, 1/1, \mathrm{i}/\mathrm{i}, \mathrm{c}/\mathrm{c}\})$ *is strictly* 3*-size-bounded funct. compl.*

(3) $\mathcal{I}_{\mathsf{br}}(\{0/0, 1/1, \mathrm{i}/\mathrm{i}\})$ *is strictly* 4*-size-bounded funct. compl.*

(4) $\mathcal{I}_{\mathsf{br}}(\{\mathrm{c}/\mathrm{c}\})$ *is strictly* 3*-size-bounded funct. compl.*

(5) $\mathcal{I}_{\mathsf{br}}(\{\mathrm{i}/0, \mathrm{i}/1\})$ *is strictly* 4*-size-bounded funct. compl.*

Proof: We assume that, for each $M \subseteq \mathcal{M}_{\mathsf{br}}$ and $k \in \mathbb{N}_1$, the restriction to $\mathcal{I}_{\mathsf{br}}(M)$ of a function ψ that witnesses k-size-bounded functional completeness of $\mathcal{I}_{\mathsf{br}}(M)$ is the identify function on $\mathcal{I}_{\mathsf{br}}(M)$ and the restriction to $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}}) \setminus \mathcal{I}_{\mathsf{br}}(M)$ has the same instruction sequence from $\mathcal{IS}_{\mathsf{br}}(M)$ as value for primitive instruction from the same equivalence class of $\mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$. It is clear that this assumption can be made without loss of generality.

Below, for each individual part of the theorem, first a function ψ that witnesses the stated size-bounded functional completeness is uniquely characterized by giving the instruction sequences for the primitive instructions for Boolean registers that are not covered by the assumption and then the strictness of the stated size-bounded functional completeness is established by checking for one of the primitive instructions concerned for which the given instruction sequence was of the greatest length that the given instruction sequence cannot be replaced by a shorter one.

We say that a $u \in \mathcal{I}_{\mathsf{br}}(\mathcal{M}_{\mathsf{br}})$ cannot be replaced by a jump instruction if there exists a $b \in \mathbb{B}$ and $n \in \mathbb{N}_1$ such that, for each $v \in \{\#l \mid l \in \mathbb{N}\}$, $|u\,;!^n| \mathbin{/\!\!/} f.BR_b^{\mathcal{M}_{\mathsf{br}}} \neq |v\,;!^n| \mathbin{/\!\!/} f.BR_b^{\mathcal{M}_{\mathsf{br}}}$.

(1) Let $M = \{0/0, 1/1, \mathrm{i}/\mathrm{i}, \mathrm{c}/\mathrm{c}, \mathrm{i}/0, \mathrm{i}/1\}$. Then instruction sequences from $\mathcal{IS}_{\mathsf{br}}(M)$ are needed for $-f.1/\mathrm{i}$ and $-f.1/\mathrm{c}$. Take ψ such that

(a) $\psi(-f.1/\mathrm{i}) = \#2$,

(b) $\psi(-f.1/\mathrm{c}) = f.\mathrm{c}/\mathrm{c} \,;\, \#2$.

Then ψ witnesses the 2-size-bounded functional completeness of $\mathcal{I}_{\mathsf{br}}(M)$. Because $-f.1/\mathrm{c}$ cannot be replaced by a jump instruction and there exists no $u \in \mathcal{I}_{\mathsf{br}}(M)$ such that $u \sim_{\mathsf{e}} -f.1/\mathrm{c}$, $\mathcal{I}_{\mathsf{br}}(M)$ is not 1-size-bounded functionally complete. Hence, $\mathcal{I}_{\mathsf{br}}(M)$ is strictly 2-size-bounded functionally complete.

(2) Let $M = \{0/0, 1/1, \mathrm{i}/\mathrm{i}, \mathrm{c}/\mathrm{c}\}$. Then instruction sequences from $\mathcal{IS}_{\mathsf{br}}(M)$ are needed for $-f.1/\mathrm{i}$, $-f.1/\mathrm{c}$, $+f.\mathrm{i}/0$, $-f.\mathrm{i}/0$, $+f.\mathrm{i}/1$, and $-f.\mathrm{i}/1$. Take ψ such that (a), (b), and

(c1) $\psi(+f.\mathrm{i}/0) = +f.\mathrm{i}/\mathrm{i} \,;\, +f.0/0 \,;\, +f.0/0$ or

(c2) $\psi(+f.\mathrm{i}/0) = -f.\mathrm{c}/\mathrm{c} \,;\, +f.0/0 \,;\, +f.0/0$ or

(c3) $\psi(+f.\mathrm{i}/0) = +f.\mathrm{i}/\mathrm{i} \,;\, +f.\mathrm{c}/\mathrm{c} \,;\, \#2$ or

(c4) $\psi(+f.\mathrm{i}/0) = -f.\mathrm{c}/\mathrm{c} \,;\, \#2 \,;\, +f.\mathrm{c}/\mathrm{c}$,

(d1) $\psi(-f.\mathrm{i}/0) = -f.\mathrm{i}/\mathrm{i} \,;\, +f.0/0 \,;\, +f.0/0$ or

(d2) $\psi(-f.\mathrm{i}/0) = +f.\mathrm{c}/\mathrm{c} \,;\, +f.0/0 \,;\, +f.0/0$ or

(d3) $\psi(-f.\mathrm{i}/0) = -f.\mathrm{i}/\mathrm{i} \,;\, \#2 \,;\, +f.\mathrm{c}/\mathrm{c}$ or

(d4) $\psi(-f.\mathrm{i}/0) = +f.\mathrm{c}/\mathrm{c} \,;\, +f.\mathrm{c}/\mathrm{c} \,;\, \#2$,

(e1) $\psi(+f.\mathrm{i}/1) = +f.\mathrm{i}/\mathrm{i} \,;\, -f.1/1 \,;\, -f.1/1$ or

(e2) $\psi(+f.\mathrm{i}/1) = -f.\mathrm{c}/\mathrm{c} \,;\, -f.1/1 \,;\, -f.1/1$ or

(e3) $\psi(+f.\mathrm{i}/1) = +f.\mathrm{i}/\mathrm{i} \,;\, \#2 \,;\, -f.\mathrm{c}/\mathrm{c}$ or

(e4) $\psi(+f.\mathrm{i}/1) = -f.\mathrm{c}/\mathrm{c} \,;\, -f.\mathrm{c}/\mathrm{c} \,;\, \#2$,

(f1) $\psi(-f.\mathrm{i}/1) = -f.\mathrm{i}/\mathrm{i} \,;\, -f.1/1 \,;\, -f.1/1$ or

(f2) $\psi(-f.\mathrm{i}/1) = +f.\mathrm{c}/\mathrm{c} \,;\, -f.1/1 \,;\, -f.1/1$ or

(f3) $\psi(-f.\mathrm{i}/1) = -f.\mathrm{i}/\mathrm{i} \,;\, -f.\mathrm{c}/\mathrm{c} \,;\, \#2$ or

(f4) $\psi(-f.\mathrm{i}/1) = +f.\mathrm{c}/\mathrm{c} \,;\, \#2 \,;\, -f.\mathrm{c}/\mathrm{c}$.[6]

[6] For several instruction sequences that start with a test instruction, there is a counterpart with the same methods in the same numbers that starts with the opposite test instruction. We refrain from mentioning these counterparts as alternatives.

Then ψ witnesses the 3-size-bounded functional completeness of $\mathcal{I}_{\mathsf{br}}(M)$. To obtain the effects of $+f.\mathrm{i/0}$, an instruction sequence from $\mathcal{IS}_{\mathsf{br}}(M)$ is needed that contains a test instruction from $\mathcal{I}_{\mathsf{br}}(M)$ with $\mathrm{i/i}$ or $\mathrm{c/c}$ as method and a primitive instruction from $\mathcal{I}_{\mathsf{br}}(M)$ with $0/0$ or $\mathrm{c/c}$ as method. Because there does not exist such an instruction sequence of length 2 with the right effects, $\mathcal{I}_{\mathsf{br}}(M)$ is not 2-size-bounded functionally complete. Hence, $\mathcal{I}_{\mathsf{br}}(M)$ is strictly 3-size-bounded functionally complete.

(3) Let $M = \{0/0, 1/1, \mathrm{i/i}\}$. Then instruction sequences from $\mathcal{IS}_{\mathsf{br}}(M)$ are needed for $-f.1/\mathrm{i}$, $+f.\mathrm{i/0}$, $-f.\mathrm{i/0}$, $+f.\mathrm{i/1}$, $-f.\mathrm{i/1}$, $f.\mathrm{c/c}$, $+f.\mathrm{c/c}$, $-f.\mathrm{c/c}$, and $-f.1/\mathrm{c}$. Take ψ such that (a), (c1) or (c3), (d1) or (d3), (e1) or (e3), (f1) or (f3), and

$$\begin{aligned}
&\text{(g)}\quad \psi(f.\mathrm{c/c}) &&= +f.\mathrm{i/i}\,;+f.0/0\,;f.1/1,\\
&\text{(h)}\quad \psi(+f.\mathrm{c/c}) &&= -f.\mathrm{i/i}\,;-f.1/1\,;+f.0/0,\\
&\text{(i)}\quad \psi(-f.\mathrm{c/c}) &&= +f.\mathrm{i/i}\,;+f.0/0\,;-f.1/1,\\
&\text{(j)}\quad \psi(-f.1/\mathrm{c}) &&= +f.\mathrm{i/i}\,;+f.0/0\,;f.1/1\,;\#2.
\end{aligned}$$

Then ψ witnesses the 4-size-bounded functional completeness of $\mathcal{I}_{\mathsf{br}}(M)$. To obtain the effects of $-f.1/\mathrm{c}$, an instruction sequence from $\mathcal{IS}_{\mathsf{br}}(M)$ is needed that contains a test instruction from $\mathcal{I}_{\mathsf{br}}(M)$ with $\mathrm{i/i}$ as method, a primitive instruction from $\mathcal{I}_{\mathsf{br}}(M)$ with $0/0$ as method, and a primitive instruction from $\mathcal{I}_{\mathsf{br}}(M)$ with $1/1$ as method. Because there does not exist such an instruction sequence of length 3 with the right effects, $\mathcal{I}_{\mathsf{br}}(M)$ is not 3-size-bounded functionally complete. Hence, $\mathcal{I}_{\mathsf{br}}(M)$ is strictly 4-size-bounded functionally complete.

(4) Let $M = \{\mathrm{c/c}\}$. Then instruction sequences from $\mathcal{IS}_{\mathsf{br}}(M)$ are needed for $-f.1/\mathrm{i}$, $-f.1/\mathrm{c}$, $+f.\mathrm{i/0}$, $-f.\mathrm{i/0}$, $+f.\mathrm{i/1}$, $-f.\mathrm{i/1}$, $f.0/0$, $+f.0/0$, $f.1/1$, $-f.1/1$, $f.\mathrm{i/i}$, $+f.\mathrm{i/i}$, and $-f.\mathrm{i/i}$. Take ψ such that (a), (b), (c4), (d4), (e4), (f4), and

$$\begin{aligned}
&\text{(k)}\quad \psi(f.0/0) &&= +f.\mathrm{c/c}\,;f.\mathrm{c/c},\\
&\text{(l)}\quad \psi(+f.0/0) &&= +f.\mathrm{c/c}\,;f.\mathrm{c/c}\,;\#2,\\
&\text{(m)}\quad \psi(f.1/1) &&= -f.\mathrm{c/c}\,;f.\mathrm{c/c},\\
&\text{(n)}\quad \psi(-f.1/1) &&= -f.\mathrm{c/c}\,;f.\mathrm{c/c}\,;\#2,
\end{aligned}$$

(o) $\psi(f.\mathrm{i}/\mathrm{i}) \quad = f.\mathrm{c}/\mathrm{c} \,;\, f.\mathrm{c}/\mathrm{c}$,

(p) $\psi(+f.\mathrm{i}/\mathrm{i}) = f.\mathrm{c}/\mathrm{c} \,;\, +f.\mathrm{c}/\mathrm{c}$,

(q) $\psi(-f.\mathrm{i}/\mathrm{i}) = f.\mathrm{c}/\mathrm{c} \,;\, -f.\mathrm{c}/\mathrm{c}$.

Then ψ witnesses the 3-size-bounded functional completeness of $\mathcal{I}_{\mathsf{br}}(M)$. To obtain the effects of $+f.0/0$, an instruction sequence from $\mathcal{IS}_{\mathsf{br}}(M)$ is needed. Because there does not exist such an instruction sequence of length 2 with the right effects, $\mathcal{I}_{\mathsf{br}}(M)$ is not 2-size-bounded functionally complete. Hence, $\mathcal{I}_{\mathsf{br}}(M)$ is strictly 3-size-bounded functionally complete.

(5) Let $M = \{\mathrm{i}/0, \mathrm{i}/1\}$. Then instruction sequences from $\mathcal{IS}_{\mathsf{br}}(M)$ are needed for $-f.1/\mathrm{i}$, $f.0/0$, $+f.0/0$, $f.1/1$, $-f.1/1$, $f.\mathrm{i}/\mathrm{i}$, $+f.\mathrm{i}/\mathrm{i}$, $-f.\mathrm{i}/\mathrm{i}$, $f.\mathrm{c}/\mathrm{c}$, $+f.\mathrm{c}/\mathrm{c}$, $-f.\mathrm{c}/\mathrm{c}$, and $-f.1/\mathrm{c}$. Take ψ such that (a) and

(r) $\psi(f.0/0) \quad = f.\mathrm{i}/0$,

(s) $\psi(+f.0/0) = f.\mathrm{i}/0 \,;\, \#2$,

(t) $\psi(f.1/1) \quad = f.\mathrm{i}/1$,

(u) $\psi(-f.1/1) = f.\mathrm{i}/1 \,;\, \#2$,

(v1) $\psi(f.\mathrm{i}/\mathrm{i}) \quad = +f.\mathrm{i}/0 \,;\, +f.\mathrm{i}/1 \,;\, -f.\mathrm{i}/0$ or

(v2) $\psi(f.\mathrm{i}/\mathrm{i}) \quad = +f.\mathrm{i}/1 \,;\, -f.\mathrm{i}/1 \,;\, +f.\mathrm{i}/0$ or

(v3) $\psi(f.\mathrm{i}/\mathrm{i}) \quad = +f.\mathrm{i}/0 \,;\, +f.\mathrm{i}/1 \,;\, \#1$ or

(v4) $\psi(f.\mathrm{i}/\mathrm{i}) \quad = +f.\mathrm{i}/1 \,;\, \#2 \,;\, +f.\mathrm{i}/0$,

(w1) $\psi(+f.\mathrm{i}/\mathrm{i}) = +f.\mathrm{i}/0 \,;\, +f.\mathrm{i}/1 \,;\, +f.\mathrm{i}/0$ or

(w2) $\psi(+f.\mathrm{i}/\mathrm{i}) = +f.\mathrm{i}/1 \,;\, -f.\mathrm{i}/1 \,;\, -f.\mathrm{i}/0$ or

(w3) $\psi(+f.\mathrm{i}/\mathrm{i}) = +f.\mathrm{i}/0 \,;\, +f.\mathrm{i}/1 \,;\, \#2$ or

(w4) $\psi(+f.\mathrm{i}/\mathrm{i}) = +f.\mathrm{i}/1 \,;\, \#2 \,;\, -f.\mathrm{i}/0$,

(x1) $\psi(-f.\mathrm{i}/\mathrm{i}) = -f.\mathrm{i}/0 \,;\, +f.\mathrm{i}/0 \,;\, +f.\mathrm{i}/1$ or

(x2) $\psi(-f.\mathrm{i}/\mathrm{i}) = -f.\mathrm{i}/1 \,;\, -f.\mathrm{i}/0 \,;\, -f.\mathrm{i}/1$ or

(x3) $\psi(-f.\mathrm{i}/\mathrm{i}) = -f.\mathrm{i}/0 \,;\, \#2 \,;\, +f.\mathrm{i}/1$ or

(x4) $\psi(-f.\mathrm{i}/\mathrm{i}) = -f.\mathrm{i}/1 \,;\, -f.\mathrm{i}/0 \,;\, \#2$,

(y1) $\psi(f.\mathrm{c}/\mathrm{c}) \quad = +f.\mathrm{i}/0 \,;\, +f.\mathrm{i}/0 \,;\, -f.\mathrm{i}/1$ or

(y2) $\psi(f.\mathrm{c}/\mathrm{c}) \quad = +f.\mathrm{i}/1 \,;\, -f.\mathrm{i}/0 \,;\, +f.\mathrm{i}/1$ or (y3) $\psi(f.\mathrm{c}/\mathrm{c}) \;\;= +f.\mathrm{i}/0 \mathbin{;} \#2 \mathbin{;} -f.\mathrm{i}/1$ or

(y4) $\psi(f.\mathrm{c}/\mathrm{c}) \;\;= +f.\mathrm{i}/1 \mathbin{;} -f.\mathrm{i}/0 \mathbin{;} \#1$,

(z1) $\psi(+f.\mathrm{c}/\mathrm{c}) = -f.\mathrm{i}/0 \mathbin{;} +f.\mathrm{i}/1 \mathbin{;} +f.\mathrm{i}/0$ or

(z2) $\psi(+f.\mathrm{c}/\mathrm{c}) = -f.\mathrm{i}/1 \mathbin{;} -f.\mathrm{i}/1 \mathbin{;} -f.\mathrm{i}/0$ or

(z3) $\psi(+f.\mathrm{c}/\mathrm{c}) = -f.\mathrm{i}/0 \mathbin{;} +f.\mathrm{i}/1 \mathbin{;} \#2$ or

(z4) $\psi(+f.\mathrm{c}/\mathrm{c}) = -f.\mathrm{i}/1 \mathbin{;} \#2 \mathbin{;} -f.\mathrm{i}/0$,

(aa1) $\psi(-f.\mathrm{c}/\mathrm{c}) = +f.\mathrm{i}/0 \mathbin{;} +f.\mathrm{i}/0 \mathbin{;} +f.\mathrm{i}/1$ or

(aa2) $\psi(-f.\mathrm{c}/\mathrm{c}) = +f.\mathrm{i}/1 \mathbin{;} -f.\mathrm{i}/0 \mathbin{;} -f.\mathrm{i}/1$ or

(aa3) $\psi(-f.\mathrm{c}/\mathrm{c}) = +f.\mathrm{i}/0 \mathbin{;} \#2 \mathbin{;} +f.\mathrm{i}/1$ or

(aa4) $\psi(-f.\mathrm{c}/\mathrm{c}) = +f.\mathrm{i}/1 \mathbin{;} -f.\mathrm{i}/0 \mathbin{;} \#2$,

(ab1) $\psi(-f.1/\mathrm{c}) = +f.\mathrm{i}/0 \mathbin{;} +f.\mathrm{i}/0 \mathbin{;} -f.\mathrm{i}/1 \mathbin{;} \#2$ or

(ab2) $\psi(-f.1/\mathrm{c}) = +f.\mathrm{i}/1 \mathbin{;} -f.\mathrm{i}/0 \mathbin{;} +f.\mathrm{i}/1 \mathbin{;} \#2$ or

(ab3) $\psi(-f.1/\mathrm{c}) = +f.\mathrm{i}/0 \mathbin{;} \#2 \mathbin{;} -f.\mathrm{i}/1 \mathbin{;} \#2$ or

(ab4) $\psi(-f.1/\mathrm{c}) = +f.\mathrm{i}/1 \mathbin{;} -f.\mathrm{i}/0 \mathbin{;} \#1 \mathbin{;} \#2$.

Then ψ witnesses the 4-size-bounded functional completeness of $\mathcal{I}_{\mathsf{br}}(M)$. To obtain the effects of $-f.1/\mathrm{c}$, an instruction sequence from $\mathcal{IS}_{\mathsf{br}}(M)$ is needed. Because there does not exist such an instruction sequence of length 3 with the right effects, $\mathcal{I}_{\mathsf{br}}(M)$ is not 3-size-bounded functionally complete. Hence, $\mathcal{I}_{\mathsf{br}}(M)$ is strictly 4-size-bounded functionally complete.

□

Theorem 3 tells us among other things that the instruction sets $\mathcal{I}_{\mathsf{br}}(\{\mathrm{c}/\mathrm{c}\})$ and $\mathcal{I}_{\mathsf{br}}(\{0/0, 1/1, \mathrm{i}/\mathrm{i}, \mathrm{c}/\mathrm{c}\})$ are both strictly 3-size-bounded functionally complete. However, the latter instruction set often gives rise to shorter instruction sequences than the former instruction set because the effects of a primitive instruction from the set $\mathcal{I}_{\mathsf{br}}(\{0/0, 1/1, \mathrm{i}/\mathrm{i}\})$ do not have to be obtained by means of two or three primitive instructions from the set $\mathcal{I}_{\mathsf{br}}(\{\mathrm{c}/\mathrm{c}\})$.

We have the following corollary of the proof of Theorem 3.

Corollary 3 *Let* $M \subset \{0/0, 1/1, \mathrm{i}/\mathrm{i}, \mathrm{c}/\mathrm{c}, \mathrm{i}/0, \mathrm{i}/1\}$. *Then:*

(1) $\mathcal{I}_{\mathsf{br}}(M)$ *is strictly* 4*-size-bounded functionally complete if* $\mathrm{c}/\mathrm{c} \notin M$ *and either* $\{0/0, 1/1, \mathrm{i}/\mathrm{i}\} \subseteq M$ *or* $\{\mathrm{i}/0, \mathrm{i}/1\} \subseteq M$*;*

(2) $\mathcal{I}_{\mathsf{br}}(M)$ *is strictly* 3*-size-bounded functionally complete if* $\mathrm{c}/\mathrm{c} \in M$*.*

7 Concluding Remarks

We have investigated instruction sequence size bounded functional completeness of instruction sets for Boolean registers. Our main results are Corollaries 2 and 3. The latter corollary covers 44 instruction sets. The covered instruction sets include the instruction sets that we used earlier in [5, 6] and many other relatively obvious instruction sets. The covered instruction sets belong to the 255 instruction sets that are non-empty subsets of one of the 256 instruction sets with the property that each possible instruction has the same effects as one from the set (see Corollary 1). It is still an open question what is the smallest k such that each of these 255 instruction sets is k-size-bounded functionally complete if it is k'-size-bounded functionally complete for some k'.

In our work on instruction sequence size complexity presented in [5], we have established several connections between instruction sequence based complexity theory and classical complexity theory. For example, we have introduced instruction sequence based counterparts of the complexity classes P/poly and NP/poly and we have formulated an instruction sequence based counterpart of the well-known complexity-theoretic conjecture that NP $\not\subseteq$ P/poly.[7] However, for many a question that arises naturally with the approach to complexity based on instruction sequence size, it is far from obvious whether a comparable question can be raised in classical complexity theory based on Turing machines or Boolean circuits. In particular, this is far from obvious for questions concerning instruction sets for Boolean registers.

Acknowledgements

We thank three anonymous referees for their helpful suggestions.

[7] The non-uniform complexity classes P/poly and NP/poly, as well as the conjecture that NP $\not\subseteq$ P/poly, are treated in many textbooks on classical complexity theory (see e.g. [1, 10, 14]).

References

[1] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, Cambridge, 2009.

[2] J. A. Bergstra and M. E. Loots. Program algebra for sequential code. *Journal of Logic and Algebraic Programming*, 51(2):125–156, 2002. doi:10.1016/S1567-8326(02)00018-8.

[3] J. A. Bergstra and C. A. Middelburg. Instruction sequence processing operators. *Acta Informatica*, 49(3):139–172, 2012. doi:10.1007/s00236-012-0154-2.

[4] J. A. Bergstra and C. A. Middelburg. *Instruction Sequences for Computer Science*, volume 2 of *Atlantis Studies in Computing*. Atlantis Press, Amsterdam, 2012. doi:10.2991/978-94-91216-65-7.

[5] J. A. Bergstra and C. A. Middelburg. Instruction sequence based nonuniform complexity classes. *Scientific Annals of Computer Science*, 24(1):47–89, 2014. doi:10.7561/SACS.2014.1.47.

[6] J. A. Bergstra and C. A. Middelburg. Instruction sequence size complexity of parity. arXiv:1412.6787v2 [cs.CC], 2014. To appear in Fundamenta Informaticae.

[7] J. A. Bergstra and C. A. Middelburg. On algorithmic equivalence of instruction sequences for computing bit string functions. *Fundamenta Informaticae*, 138(4):411–434, 2015.doi:10.3233/FI-2015-1219.

[8] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification I: Equations and Initial Semantics*, volume 6 of *EATCS Monographs*. Springer-Verlag, Berlin, 1985. doi:10.1007/978-3-642-69962-7.

[9] W. A. Hodges. *Model Theory*, volume 42 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, Cambridge, 1993.

[10] S. Homer and A. L. Selman. *Computability and Complexity Theory*. Springer-Verlag, Berlin, second edition, 2011. doi:10.1007/978-1-4614-0682-2.

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

[12] D. Sannella and A. Tarlecki. Algebraic preliminaries. In E. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, pages 13–30. Springer-Verlag, Berlin, 1999. doi:10.1007/978-3-642-59851-7_2.

[13] D. Sannella and A. Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Monographs in Theoretical Computer Science, An EATCS Series. Springer-Verlag, Berlin, 2012. doi:10.1007/978-3-642-17336-3.

[14] I. Wegener. *Complexity Theory – Exploring the Limits of Efficient Algorithms*. Springer-Verlag, Berlin, 2005. doi:10.1007/3-540-27477-4.

[15] M. Wirsing. Algebraic specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 675–788. Elsevier, Amsterdam, 1990.