



UvA-DARE (Digital Academic Repository)

A Hoare-Like Logic of Asserted Single-Pass Instruction Sequences

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

DOI

[10.7561/SACS.2016.2.125](https://doi.org/10.7561/SACS.2016.2.125)

Publication date

2016

Document Version

Final published version

Published in

Scientific Annals of Computer Science

License

Unspecified

[Link to publication](#)

Citation for published version (APA):

Bergstra, J. A., & Middelburg, C. A. (2016). A Hoare-Like Logic of Asserted Single-Pass Instruction Sequences. *Scientific Annals of Computer Science*, 26(2), 125-156. <https://doi.org/10.7561/SACS.2016.2.125>

General rights

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible.

A Hoare-Like Logic of Asserted Single-Pass Instruction Sequences

J.A. BERGSTRA¹, C.A. MIDDELBURG¹

Abstract

We present a formal system for proving the partial correctness of a single-pass instruction sequence as considered in program algebra by decomposition into proofs of the partial correctness of segments of the single-pass instruction sequence concerned. The system is similar to Hoare logics, but takes into account that, by the presence of jump instructions, segments of single-pass instruction sequences may have multiple entry points and multiple exit points. It is intended to support a sound general understanding of the issues with Hoare-like logics for low-level programming languages.

Keywords: Hoare logic, asserted single-pass instruction sequence, soundness, completeness in the sense of Cook.

1 Introduction

In [15], Hoare introduced a kind of formal system for proving the partial correctness of a program by decomposition into proofs of the partial correctness of segments of the program concerned. Formal systems of this kind are now known as Hoare logics. The programs considered in [15] are programs in a simple high-level programming language without goto statements. Hoare logics for this simple high-level programming language and extensions of it without goto statements have been extensively studied since (see e.g. [8, 9, 11] for individual studies and [1] for a survey). Hoare logics and Hoare-like

¹Informatics 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.

logics for simple high-level programming languages with goto statements have been studied since as well (see e.g. [12, 10, 25]).

Work on Hoare-like logics for low-level programming languages started only recently. All the work that we know of takes ad hoc restrictions and features of machine- or assembly-level programs into account (see e.g. [19]) or abstracts in an ad hoc way from instruction sequences as found in low-level programs (see e.g. [21]). We consider it important for a sound understanding of the issues in this area to give consideration to generality and faithfulness of abstraction instead. This is what motivated us to do the work presented in this paper.

We present a Hoare-like logic for single-pass instruction sequences as considered in [2]. The instruction sequences in question are finite or eventually periodic infinite sequences of instructions of which each instruction is executed at most once and can be dropped after it has been executed or jumped over. We will come back to the choice for those instruction sequences. The presented Hoare-like logic has to take into account that, by the presence of jump instructions, segments of instruction sequences may have multiple entry points and multiple exit points. Because of this, it is closer to the inductive assertion method for program flowcharts introduced by Floyd in [14] than most other Hoare and Hoare-like logics.

The asserted programs of the form $\{P\}S\{Q\}$ of Hoare logics are replaced in the presented Hoare-like logic by asserted instruction sequences of the form $\{b:P\}S\{e:Q\}$, where b and e are a positive natural number and a natural number, respectively. P and Q are regular pre- and post-conditions. That is, they concern the input-output behaviour of the instruction sequence segment S . Loosely speaking, b represents the additional pre-condition that execution enters the instruction sequence segment S at its b th instruction and, if e is positive, e represents the additional post-condition that either execution exits the instruction sequence segment S by going to the e th instruction following it or becomes inactive in S . In the case that e equals zero, e represents the additional post-condition that execution either terminates or becomes inactive in S (instruction sequences with explicit termination instructions are considered).

The form of the asserted instruction sequences is inspired by [25]. However, under the interpretation of [25], e would represent the additional post-condition that execution reaches the e th instruction following the first instruction of the instruction sequence segment concerned. Because this may be an instruction before the first instruction following the segment,

this interpretation allows of asserted instruction sequences that concern the internals of the segment. For this reason, we consider this interpretation not conducive to compositional proofs.

In other related work, e.g. in [21], the additional pre- and post-condition represented by b and e must be explicitly formulated and conjoined with the regular pre- and post-condition, respectively. This alternative reduces the conciseness of pre- and post-conditions considerably. Moreover, an effect ensuing from this alternative is that assertions can be formulated in which aspects of input-output behaviour and flow of execution are combined in ways that are unnecessary for proving partial correctness. For these reasons, we decided not to opt for this alternative.

There is a tendency in work on Hoare-like logics to use a separation logic instead of classical first-order logic for pre- and post-conditions to deal with programs that alter data structures (see e.g. [20]). This tendency is also found in work on Hoare-like logics for low-level programming languages (see e.g. [17]). Because our intention is to present a Hoare-like logic that supports a sound general understanding of the issues with Hoare-like logics for low-level programming languages, we believe that we should stick to classical first-order logic until it proves to be inadequate. This is the reason why classical first-order logic is used for pre- and post-conditions in this paper.

As mentioned before, the presented Hoare-like logic concerns single-pass instruction sequences as considered in [2]. 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 [2], and of which an enumeration is available at [18], is to understand the concept of a program. The basis of all this work is an algebraic theory of single-pass instruction sequences, called program algebra, and an algebraic theory of mathematical objects that represent in a direct way the behaviours produced by instruction sequences under execution, called basic thread algebra.² 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

²In [2], basic thread algebra is introduced under the name basic polarized process algebra.

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, this work 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, garbage collection, program parallelization for the purpose of explicit multi-threading and virus detection.

Judging by our experience gained in the work referred to above, we think that generality and faithfulness of abstraction are well taken into consideration in a Hoare-like logic for single-pass instruction sequences as considered in [2]. This explains the choice for those instruction sequences. As in the work referred to above, the work presented in this paper is carried out in the setting of program algebra and basic thread algebra.

This paper is organized as follows. First, we give a survey of program algebra and basic thread algebra (Section 2) and a survey of the extension of basic thread algebra that is used in this paper (Section 3). Next, we present a Hoare-like logic of asserted single-pass instruction sequences (Section 4), give an example of its use (Section 5), and show that it is sound and complete in the sense of Cook (Section 6). Finally, we make some concluding remarks (Section 7).

Some familiarity with algebraic specification is assumed in this paper. The relevant notions are explained in handbook chapters and books on algebraic specification, e.g. [13, 22, 23, 26].

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. A comprehensive introduction to what is surveyed in the preliminary sections can among other things be found in [5].

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 [6].

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 f and t . 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{J} 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 t 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, execution becomes inactive;
- 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 t is produced;
- the effect of a forward jump instruction $\#l$ is that execution proceeds with the l th next primitive instruction — if l equals 0 or there is no primitive instruction to proceed with, execution becomes inactive;
- the effect of the termination instruction $!$ is that execution terminates.

Execution becomes inactive if no more basic instructions are executed, but execution does not terminate.

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 $u : \rightarrow \mathbf{IS}$;
- the binary *concatenation* operator $_ ; _ : \mathbf{IS} \times \mathbf{IS} \rightarrow \mathbf{IS}$;
- the unary *repetition* operator $_^\omega : \mathbf{IS} \rightarrow \mathbf{IS}$.

Terms of sort **IS** are built as usual in the one-sorted case.³ 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. Hence, taking these notational conventions into account, the syntax of closed terms of sort **IS** can be defined in Backus-Naur style as follows:

$$CT ::= a \mid +a \mid -a \mid \#l \mid ! \mid CT ; CT \mid CT^\omega ,$$

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

A closed PGA term is considered to denote a non-empty, finite or eventually periodic infinite sequence of primitive instructions.⁴ The instruction sequence denoted by a closed term of the form $t ; t'$ 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. A simple example of a closed PGA term is

$$(+a ; \#2 ; \#3 ; b ; !)^\omega .$$

On execution of the instruction sequence denoted by this term, first the basic instruction a is executed repeatedly until its execution produces the reply t , next the basic instruction b is executed, and after that execution terminates.

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 positive natural

³Notice that all PGA term are of sort **IS**.

⁴An 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

number. For each natural number n , the term t^n , where t is a PGA term, is defined by induction on n as follows: $t^0 = \#0$, $t^1 = t$, and $t^{n+2} = t ; t^{n+1}$. Some simple examples of equations derivable from the axioms of PGA are

$$(a ; b)^\omega ; ! = a ; (b ; a)^\omega ,$$

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

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 $^\omega$ defined as follows:
 - if U is a finite sequence over \mathfrak{I} , then U^ω 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 an infinite sequence over \mathfrak{I} , then U^ω is U .

We confine ourselves to this model of PGA, which is an initial 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 and write $len(t)$, where t is a closed PGA term denoting a finite PGA instruction sequence, for the length of the PGA instruction sequence denoted by t . We stipulate that $len(t) = \omega$ if t is a closed PGA term denoting an infinite instruction sequence, where $n < \omega$ for all $n \in \mathbb{N}$.

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 f and t .

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 $D : \rightarrow \mathbf{T}$;
- the *termination* constant $S : \rightarrow \mathbf{T}$;
- for each $a \in \mathcal{A}$, the binary *postconditional composition* operator $-\trianglelefteq a \triangleright-$: $\mathbf{T} \times \mathbf{T} \rightarrow \mathbf{T}$.

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 $t \trianglelefteq a \triangleright t$. 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 $t \trianglelefteq a \triangleright t'$ will first perform a , and then proceed as the thread denoted by t if the reply from the execution environment is t and proceed as the thread denoted by t' if the reply from the execution environment is f . The thread denoted by S will do no more than terminate and the thread denoted by D will become inactive. A simple example of a closed BTA term is

$$(b \circ S) \trianglelefteq a \triangleright D .$$

This term denotes the thread that first performs basic action a , if the reply from the execution environment on performing a is t , next performs the basic action b and after that terminates, and if the reply from the execution environment on performing a is f , next becomes 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. [4]). We are only interested in models of BTA in which

Table 2: Axioms for the thread extraction operator

$ a = a \circ \mathbf{D}$	$ \#l = \mathbf{D}$
$ a; X = a \circ X $	$ \#0; X = \mathbf{D}$
$ +a = a \circ \mathbf{D}$	$ \#1; X = X $
$ +a; X = X \trianglelefteq a \triangleright \#2; X $	$ \#l + 2; u = \mathbf{D}$
$ -a = a \circ \mathbf{D}$	$ \#l + 2; u; X = \#l + 1; X $
$ -a; X = \#2; X \trianglelefteq a \triangleright X $	$ \! = \mathbf{S}$
	$ \! ; X = \mathbf{S}$

sets of recursion equations have unique solutions, such as the projective limit model of BTA presented in [5]. 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. The behaviours produced by PGA instruction sequences under execution are exactly the behaviours represented by regular threads, with the basic instructions taken for basic actions. The behaviours produced by finite PGA instruction sequences under execution are the behaviours represented by finite threads.

We combine PGA with BTA and extend the combination with the *thread extraction* operator $|-| : \mathbf{IS} \rightarrow \mathbf{T}$, the axioms given in Table 2, and the rule that $|X| = \mathbf{D}$ if X has an infinite chain of forward jumps beginning at its first primitive instruction.⁵ In Table 2, a stands for an arbitrary basic instruction from \mathfrak{A} , u stands for an arbitrary primitive instruction from \mathfrak{J} , and l stands for an arbitrary natural number from \mathbb{N} . For each closed PGA term t , $|t|$ denotes the behaviour produced by the instruction sequence denoted by t under execution.

A simple example of thread extraction is

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

In the case of infinite instruction sequences, thread extraction yields threads

⁵This rule, which can be formalized using an auxiliary structural congruence predicate (see e.g. [3]), is unnecessary when considering only finite PGA instruction sequences.

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

$$x = (b \circ \mathbf{S}) \trianglelefteq a \triangleright x .$$

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. The processing of a method may involve a change of the service. At completion of the processing of a method, the service produces a reply value. For example, a service may be able to process methods for pushing a natural number on a stack (**push**: n), popping the top element from the stack (**pop**), and testing whether the top element of the stack equals a natural number (**topeq**: n). Processing of a pushing method or a popping method changes the service and produces the reply value **t** if no stack underflow occurs and **f** otherwise. Processing of a testing method does not change the service and produces the reply value **t** if the test succeeds and **f** otherwise.

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 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 extend BTA with services, service families, a composition operator for service families, and an operator that is concerned with this kind of interaction. This section originates from [4].

In SFA, the algebraic theory of service families introduced below, 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 **S** of *services*;
 - the sort **R** of *replies*;

and the following constants and operators:

- the *empty service* constant $\delta : \rightarrow \mathbf{S}$;
- the *reply* constants $\mathbf{f}, \mathbf{t}, \mathbf{d} : \rightarrow \mathbf{R}$;
- for each $m \in \mathcal{M}$, the *derived service* operator $\frac{\partial}{\partial m} : \mathbf{S} \rightarrow \mathbf{S}$;
- for each $m \in \mathcal{M}$, the *service reply* operator $\varrho_m : \mathbf{S} \rightarrow \mathbf{R}$;
- a minimal $\Sigma_{\mathcal{S}}$ -algebra \mathcal{S} has been given in which \mathbf{f} , \mathbf{t} , and \mathbf{d} are mutually different, and
 - $\bigwedge_{m \in \mathcal{M}} \frac{\partial}{\partial m}(z) = z \wedge \varrho_m(z) = \mathbf{d} \Rightarrow z = \delta$ holds;
 - for each $m \in \mathcal{M}$, $\frac{\partial}{\partial m}(z) = \delta \Leftrightarrow \varrho_m(z) = \mathbf{d}$ holds.

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

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

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

The actual services could, for example, be the natural number stack services sketched at the beginning of this section. In that case, we take the set $\{NNS_\sigma \mid \sigma \in \mathbb{N}^*\}$ of *natural number stack services* as the set \mathcal{S} of services and, for each $m \in \mathcal{M}$, we take the functions $\frac{\partial}{\partial m}$ and ϱ_m such that $(n, n' \in \mathbb{N}, \sigma \in \mathbb{N}^*)$:⁶

$$\begin{array}{ll}
 \frac{\partial}{\partial \text{push}:n}(NNS_\sigma) = NNS_{n\sigma}, & \varrho_{\text{push}:n}(NNS_\sigma) = \mathbf{t}, \\
 \frac{\partial}{\partial \text{pop}}(NNS_{n'\sigma}) = NNS_\sigma, & \varrho_{\text{pop}}(NNS_{n'\sigma}) = \mathbf{t}, \\
 \frac{\partial}{\partial \text{pop}}(NNS_\epsilon) = NNS_\epsilon, & \varrho_{\text{pop}}(NNS_\epsilon) = \mathbf{f}, \\
 \frac{\partial}{\partial \text{topeq}:n}(NNS_{n'\sigma}) = NNS_{n'\sigma}, & \varrho_{\text{topeq}:n}(NNS_{n'\sigma}) = \mathbf{t} \text{ if } n = n', \\
 & \varrho_{\text{topeq}:n}(NNS_{n'\sigma}) = \mathbf{f} \text{ if } n \neq n', \\
 \frac{\partial}{\partial \text{topeq}:n}(NNS_\epsilon) = NNS_\epsilon, & \varrho_{\text{topeq}:n}(NNS_\epsilon) = \mathbf{f}, \\
 \frac{\partial}{\partial m}(NNS_\sigma) = \delta \text{ if } m \notin \mathcal{M}_{NNS}, & \varrho_m(NNS_\sigma) = \mathbf{d} \text{ if } m \notin \mathcal{M}_{NNS},
 \end{array}$$

where $\mathcal{M}_{NNS} = \{\text{push}:n \mid n \in \mathbb{N}\} \cup \{\text{pop}\} \cup \{\text{topeq}:n \mid n \in \mathbb{N}\}$.

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.

⁶We write ϵ for the empty sequence and $n\sigma$ for the sequence σ with n prepended to it.

Table 3: Axioms of SFA

$u \oplus \emptyset = u$	SFC1	$\partial_F(\emptyset) = \emptyset$	SFE1
$u \oplus v = v \oplus u$	SFC2	$\partial_F(f.z) = \emptyset$ if $f \in F$	SFE2
$(u \oplus v) \oplus w = u \oplus (v \oplus w)$	SFC3	$\partial_F(f.z) = f.z$ 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

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

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

We assume that there are infinitely many variables of sort **S**, including z , and infinitely many variables of sort **SF**, including u, v, w . Terms are built as usual in the many-sorted case (see e.g. [22, 26]). 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, \dots, t_n are terms of sort **SF**, for the term $t_1 \oplus \dots \oplus t_n$.

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 the name of this service is 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.

Table 4: Axioms for the apply operator

$\mathbf{S} \bullet u = u$	A1
$\mathbf{D} \bullet u = \emptyset$	A2
$(x \trianglelefteq f.m \trianglerighteq y) \bullet \partial_{\{f\}}(u) = \emptyset$	A3
$(x \trianglelefteq f.m \trianglerighteq y) \bullet (f.t \oplus \partial_{\{f\}}(u)) = x \bullet (f.\frac{\partial}{\partial m}t \oplus \partial_{\{f\}}(u))$ if $\varrho_m(t) = \mathbf{t}$	A4
$(x \trianglelefteq f.m \trianglerighteq y) \bullet (f.t \oplus \partial_{\{f\}}(u)) = y \bullet (f.\frac{\partial}{\partial m}t \oplus \partial_{\{f\}}(u))$ if $\varrho_m(t) = \mathbf{f}$	A5
$(x \trianglelefteq f.m \trianglerighteq y) \bullet (f.t \oplus \partial_{\{f\}}(u)) = \emptyset$ if $\varrho_m(t) = \mathbf{d}$	A6

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

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

- the binary *apply* operator $_ \bullet _ : \mathbf{T} \times \mathbf{SF} \rightarrow \mathbf{SF}$;

and the axioms given in Table 4. In this table, 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 apply if inappropriate foci or methods are involved. We use infix notation for the apply operator.

The service family denoted by a closed term of the form $t \bullet t'$ is the service family that results 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.

In the case of the stack services described earlier in this section, the following two equations are simple examples of derivable equations:

$$\begin{aligned} ((\mathbf{nns}.\mathbf{pop} \circ \mathbf{S}) \trianglelefteq \mathbf{nns}.\mathbf{topeq};0 \trianglerighteq \mathbf{S}) \bullet \mathbf{nns}.\mathbf{NNS}_{0\sigma} &= \mathbf{nns}.\mathbf{NNS}_{\sigma} , \\ ((\mathbf{nns}.\mathbf{pop} \circ \mathbf{S}) \trianglelefteq \mathbf{nns}.\mathbf{topeq};0 \trianglerighteq \mathbf{S}) \bullet \mathbf{nns}.\mathbf{NNS}_{1\sigma} &= \mathbf{nns}.\mathbf{NNS}_{1\sigma} . \end{aligned}$$

4 Hoare-Like Logic for PGA⁷

In this section, we introduce a formal system for proving the partial correctness of instruction sequences as considered in PGA. Unlike segments of programs written in the high-level programming languages for which Hoare logics have been developed, segments of single-pass instruction sequences may have multiple entry points and multiple exit points. Therefore, the asserted programs of the form $\{P\}S\{Q\}$ of Hoare logics fall short in the case of single-pass instruction sequences. The formulas in the formal system introduced here will be called asserted instruction sequences.

We will look upon foci as (program) variables. This is justified by the fact that foci are names of objects that may be modified on execution of an instruction sequence. The objects concerned are services. What is assumed here with respect to services is the same as in Section 3. This means that a signature $\Sigma_{\mathcal{S}}$ that includes specific sorts, constants and operators and a minimal $\Sigma_{\mathcal{S}}$ -algebra \mathcal{S} that satisfies specific conditions have been given.

In the formal system introduced here, classical first-order logic with equality is used for pre- and post-conditions. The particular choice of logical constants, connectives and quantifiers does not matter. However, for convenience, it is assumed that the following is included: (a) the constants \top (for truth) and F (for falsity), (b) the connectives \neg (for negation), \wedge (for conjunction), \vee (for disjunction), and \Rightarrow (for implication), (c) the quantifiers \forall (for universal quantification) and \exists (for existential quantification).

We write $\mathcal{L}_{\mathcal{S}}$ for the many-sorted first-order language with equality over the signature $\Sigma_{\mathcal{S}}$ where free variables of sort \mathbf{S} belong to the set \mathcal{F} . Moreover, we write $\mathcal{C}_{\mathbf{IS}}$ for the set of all closed terms of sort \mathbf{IS} in the case where the set $\{f.m \mid f \in \mathcal{F}, m \in \mathcal{M}\}$ is taken as the set \mathfrak{A} of basic instructions.

An *asserted instruction sequence* is a formula of the form $\{b:P\}S\{e:Q\}$, where $S \in \mathcal{C}_{\mathbf{IS}}$, $P, Q \in \mathcal{L}_{\mathcal{S}}$, $b \in \mathbb{N}^+$, and $e \in \mathbb{N}$.⁸ The intuitive meaning of an asserted instruction sequence $\{b:P\}S\{e:Q\}$ is as follows:

- if $b \leq \text{len}(S)$ and $e > 0$, the intuitive meaning is:

if execution enters the instruction sequence segment S at its b th instruction and P holds when execution enters S , then either execution becomes inactive in S or execution exits

⁷The term “Hoare-like logic”, which stands for “logic like Hoare” if taken literally, is widely used since 1981 with the meaning “logic like Hoare logic” and we conform to this usage.

⁸We write \mathbb{N}^+ for the set $\{n \in \mathbb{N} \mid n > 0\}$.

S by going to the e th instruction following S and Q holds when execution exits S ;

- if $b \leq \text{len}(S)$ and $e = 0$, the intuitive meaning is:

if execution enters the instruction sequence segment S at its b th instruction and P holds when execution enters S , then either execution becomes inactive in S or execution terminates in S and Q holds when execution terminates in S ;⁹

- if $b > \text{len}(S)$, an intuitive meaning is lacking.

For convenience, we did not exclude the case where $b > \text{len}(S)$. Instead, we made the choice that any asserted instruction sequence $\{b: P\} S \{e: Q\}$ with $b > \text{len}(S)$ does not hold (irrespective of the choice of \mathcal{S}).

Before we make precise what it means that an asserted instruction sequence holds in \mathcal{S} , we introduce some special terminology and notation.

In the setting of PGA, what we mean by a *state* is a function from a finite subset of \mathcal{F} to the interpretation of sort \mathbf{S} in \mathcal{S} . Let $F \subset \mathcal{F}$ be such that F is finite. Then a *state representing term for F with respect to \mathcal{S}* is a closed term t of sort \mathbf{SF} for which, for all $f \in F$, $\partial_{\{f\}}(t) = t$ does not hold in the free extension of \mathcal{S} to a model of SFA. Notice that $\partial_{\{f\}}(t) = t$ does not hold iff the interpretation of t is a service family to which a service with name f belongs. Let $P \in \mathcal{L}_{\mathcal{S}}$, and let F' be the set all foci that belong to the free variables of P . Then a *state representing term for P with respect to \mathcal{S}* is a closed term t of sort \mathbf{SF} that is a state representing term for F' with respect to \mathcal{S} . Let $S \in \mathcal{C}_{\mathbf{IS}}$, and let F'' be the set all foci that occur in S . Then a *state representing term for S with respect to \mathcal{S}* is a closed term t of sort \mathbf{SF} that is a state representing term for F'' with respect to \mathcal{S} .

We write $P[t]$, where t is a state representing term for P with respect to \mathcal{S} , for P with, for each $f \in \mathcal{F}$, all free occurrences of f replaced by a closed term t' of sort \mathbf{S} such that $t = f.t' \oplus \partial_{\{f\}}(t)$ holds in the free extension of \mathcal{S} to a model of SFA. Thus, the interpretation of the term t' replacing the free occurrences of f is the service associated with f in the state represented by t . Notice that an equation between terms of sort \mathbf{SF} holds in the free extension of \mathcal{S} to a model of SFA iff it is derivable from the axioms of SFA.

⁹Recall that execution becomes inactive if no more basic instructions are executed, but execution does not terminate.

We write $|S|_{b,0}$ for $|\#b; S|$ and $|S|_{b,e}$, where $e > 0$, for $|\#b; S; \sigma(e)|$ where, for each $n > 0$, $\sigma(n)$ is defined by induction on n as follows: $\sigma(1) = !$ and $\sigma(n+1) = \#0; \sigma(n)$. In the case where $b \leq \text{len}(S) \leq \omega$ and $e > 0$, the thread denoted by $|S|_{b,e}$ represents the behaviour that differs from the behaviour produced by the instruction sequence segment S in isolation if execution enters the segment at its b th instruction only by terminating instead of becoming inactive if execution exits the segment by going to the e th instruction following it. This adaptation of the behaviour is a technicality by which it is possible to obtain the state at the time that execution exits the segment by means of the apply operation \bullet .

An asserted instruction sequence $\{b : P\} S \{e : Q\}$ holds in \mathcal{S} , written $\mathcal{S} \models \{b : P\} S \{e : Q\}$, if $b \leq \text{len}(S)$ and for all closed terms t and t' of sort \mathbf{SF} that are state representing terms for P , Q , and S with respect to \mathcal{S} :

$$\begin{aligned} \mathcal{S} \models P[t] \text{ implies } \mathcal{M}_{\mathcal{S}} \models |S|_{b,e'} \bullet t = \emptyset \text{ for all } e' \in \mathbb{N} \text{ with } e \neq e' \\ \text{and} \\ \mathcal{S} \models P[t] \text{ and } \mathcal{M}_{\mathcal{S}} \models |S|_{b,e} \bullet t = t' \text{ imply } \mathcal{S} \models Q[t'], \end{aligned}$$

where $\mathcal{M}_{\mathcal{S}}$ is the model of the combination of PGA, BTA, and SFA extended with the thread extraction operator, the apply operator, and the axioms for these operators such that the restrictions to the signatures of PGA, BTA, and SFA are the initial model of PGA, the projective limit model of BTA, and the free extension of \mathcal{S} to a model of SFA, respectively. The existence of such a model follows from the fact that the signatures of PGA, BTA, and SFA are disjoint by the amalgamation result about expansions presented as Theorem 6.1.1 in [16] (adapted to the many-sorted case). The occurrences of \mathcal{S} in the above definition can be replaced by $\mathcal{M}_{\mathcal{S}}$.

Notice that for all $S \in \mathcal{C}_{\mathbf{IS}}$, $Q \in \mathcal{L}_{\mathcal{S}}$, $b \in \mathbb{N}^+$ with $b \leq \text{len}(S)$, and $e \in \mathbb{N}$, $\mathcal{S} \models \{b : \mathbf{F}\} S \{e : Q\}$. However, there exist $S \in \mathcal{C}_{\mathbf{IS}}$, $P \in \mathcal{L}_{\mathcal{S}}$, $b \in \mathbb{N}^+$ with $b \leq \text{len}(S)$, and $e \in \mathbb{N}$ such that $\mathcal{S} \not\models \{b : P\} S \{e : \mathbf{T}\}$. This is the case because, if execution enters the instruction sequence segment S at its b th instruction and P holds when execution enters S , then there may be no unique way in which execution exits S and, if there is a unique way, it may be by going to another than the e th instruction following S .

We could have dealt with the above-mentioned non-uniqueness by supporting multiple exit points in asserted instruction sequences. In that case, we would have asserted instruction sequences of the form $\{b : P\} S \{e_1, \dots, e_n : Q\}$ satisfying $\mathcal{S} \models \{b : P\} S \{e_1, \dots, e_n : Q\}$ iff $\mathcal{S} \models \{b : P\} S \{e_i : Q\}$ for all $i \in \{1, \dots, n\}$. This means that it is sufficient to add to the axioms and rules of inference of our Hoare-like logic (introduced below) the rules of inference

corresponding to this equivalence. These additional rules are such that nothing gets lost if $\{b : P\} S \{e_1, \dots, e_n : Q\}$ is simply considered a shorthand for the set $\{\{b : P\} S \{e_i : Q\} \mid i \in \{1, \dots, n\}\}$ of asserted instruction sequences.

The axioms and rules of inference of our Hoare-like logic of asserted single-pass instruction sequences are given in Table 5. In this table, S, S_1, S_2 stand for arbitrary closed terms from $\mathcal{C}_{\mathbf{IS}}$, $P, P', P_1, P_2, \dots, Q, Q', Q_1, Q_2, \dots$, and R stand for arbitrary formulas from $\mathcal{L}_{\mathbf{S}}$, b, b_1, b_2, \dots stand for arbitrary positive natural numbers, e, i stand for arbitrary natural numbers, x, y stand for arbitrary variables of some sort in $\Sigma_{\mathbf{S}}$, f stands for an arbitrary focus from \mathcal{F} , and m stands for an arbitrary method from \mathcal{M} . Moreover, $\text{var}(P)$ denotes the set all foci that belong to the free variables of P and $\text{var}(S)$ denotes the set of all foci that occur in S . We write $\Psi \vdash' \phi$, where Ψ is a finite set of asserted instruction sequences and ϕ is an asserted instruction sequence, for provability of ϕ from Ψ without applications of the repetition rule (R5).

The axioms concern the smallest instruction sequence segments, namely single instructions. Axioms A1–A8 are similar to the assignment axiom found in most Hoare logics. They are somewhat more complicated than the assignment axiom because they concern instructions that may cause execution to become inactive and, in case of axioms A3–A8, instructions that have two exit points. Axioms A9–A11, which concern jump instructions and the termination instruction, are very simple and speak for themselves.

Concatenation needs four rules because instruction sequence segments may be prefixed or suffixed by redundant instruction sequence segments in several ways. Rule R1 concerns the obvious case, namely the case where execution enters the whole by entering the first instruction sequence segment and execution exits the whole by exiting the second instruction sequence segment. Rule R2 concerns the case where execution exits the whole by exiting the first instruction sequence segment. Rule R3 concerns the case where execution becomes inactive or terminates in the whole by doing so in the first instruction sequence segment. Rule R4 concerns the case where execution enters the whole by entering the second instruction sequence segment.

The repetition rule (rule R5) is reminiscent of the recursion rule found in Hoare logics for high-level programming languages that covers calls of (parameterless) recursive procedures (see e.g. [1]). This rule is actually a rule schema: there is an instance of this rule for each $k, n > 0$ with $k \leq n$. In many cases, the instance for $k = 1$ and $n = 1$ suffices. The need for the rules R6–R9 is not clear at first sight, but without them the presented formal system would be incomplete. Although these rules do not explicitly deal

Table 5: Hoare-Like Logic of Asserted Single-Pass Instruction Sequences

BASIC INSTRUCTION AXIOMS:

A1 : $\{1 : \varrho_m(f) \neq d \wedge P[\frac{\partial}{\partial m}(f)/f]\} f.m \{1 : P\}$

A2 : $\{1 : \varrho_m(f) = d\} f.m \{0 : F\}$

POSITIVE TEST INSTRUCTION AXIOMS:

A3 : $\{1 : \varrho_m(f) = t \wedge P[\frac{\partial}{\partial m}(f)/f]\} +f.m \{1 : P\}$

A4 : $\{1 : \varrho_m(f) = f \wedge P[\frac{\partial}{\partial m}(f)/f]\} +f.m \{2 : P\}$

A5 : $\{1 : \varrho_m(f) = d\} +f.m \{0 : F\}$

NEGATIVE TEST INSTRUCTION AXIOMS:

A6 : $\{1 : \varrho_m(f) = t \wedge P[\frac{\partial}{\partial m}(f)/f]\} -f.m \{2 : P\}$

A7 : $\{1 : \varrho_m(f) = f \wedge P[\frac{\partial}{\partial m}(f)/f]\} -f.m \{1 : P\}$

A8 : $\{1 : \varrho_m(f) = d\} -f.m \{0 : F\}$

FORWARD JUMP INSTRUCTION AXIOMS:

A9 : $\{1 : P\} \#i+1 \{i+1 : P\}$ A10 : $\{1 : T\} \#0 \{0 : F\}$

TERMINATION INSTRUCTION AXIOM:

A11 : $\{1 : P\} ! \{0 : P\}$

CONCATENATION RULES:

R1 : $\frac{\{b : P\} S_1 \{i : Q\}, \{i : Q\} S_2 \{e : R\}}{\{b : P\} S_1 ; S_2 \{e : R\}} \quad i > 0$

R2 : $\frac{\{b : P\} S_1 \{e + \text{len}(S_2) : Q\}}{\{b : P\} S_1 ; S_2 \{e : Q\}} \quad e > 0$ R3 : $\frac{\{b : P\} S_1 \{0 : Q\}}{\{b : P\} S_1 ; S_2 \{0 : Q\}}$

R4 : $\frac{\{b : P\} S_2 \{e : Q\}}{\{b + \text{len}(S_1) : P\} S_1 ; S_2 \{e : Q\}}$

REPETITION RULE (for each $k, n > 0$ with $k \leq n$):

$$\{b_1 : P_1\} S^\omega \{0 : Q_1\}, \dots, \{b_n : P_n\} S^\omega \{0 : Q_n\} \vdash' \{b_1 : P_1\} S ; S^\omega \{0 : Q_1\}$$

R5 :

$$\frac{\{b_1 : P_1\} S^\omega \{0 : Q_1\}, \dots, \{b_n : P_n\} S^\omega \{0 : Q_n\} \vdash' \{b_n : P_n\} S ; S^\omega \{0 : Q_n\}}{\{b_k : P_k\} S^\omega \{0 : Q_k\}}$$

ALTERNATIVES RULE:

R6 : $\frac{\{b : P\} S \{e : R\}, \{b : Q\} S \{e : R\}}{\{b : P \vee Q\} S \{e : R\}}$

INVARIANCE RULE:

R7 : $\frac{\{b : P\} S \{e : Q\}}{\{b : P \wedge R\} S \{e : Q \wedge R\}} \quad \text{var}(R) \cap \text{var}(S) = \emptyset$

Table 5: (Continued)

ELIMINATION RULE:

$$\text{R8: } \frac{\{b : P\} S \{e : Q\}}{\{b : \exists x \bullet P\} S \{e : Q\}} \{x\} \cap (\text{var}(S) \cup \text{var}(Q)) = \emptyset$$

SUBSTITUTION RULE:

$$\text{R9: } \frac{\{b : P\} S \{e : Q\}}{\{b : P[y/x]\} S \{e : Q[y/x]\}} \{x\} \cap \text{var}(S) = \emptyset, \{y\} \cap \text{var}(S) = \emptyset$$

CONSEQUENCE RULE:

$$\text{R10: } \frac{P \Rightarrow P', \{b : P'\} S \{e : Q'\}, Q' \Rightarrow Q}{\{b : P\} S \{e : Q\}}$$

with repetition, they would not be needed for completeness in the absence of repetition.

The consequence rule (rule R10) is found in one form or another in all Hoare logics and Hoare-like logics. This rule allows to make use of formulas from $\mathcal{L}_{\mathcal{S}}$ that hold in \mathcal{S} to strengthen pre-conditions and weaken post-conditions.

Because there is no rule of inference to deal with nested repetitions, it seems at first sight that we cannot have a completeness result for the presented Hoare-like logic. However, a closer look at this matter yields something different. The crux is that the following rule of inference is derivable from rules R3 and R5:

$$\frac{\{b : P\} S \{0 : Q\}}{\{b : P\} S^\omega \{0 : Q\}}.$$

We have the following result:

Theorem 1 *Let $\text{Th}(\mathcal{S})$ be the set of all formulas of $\mathcal{L}_{\mathcal{S}}$ that hold in \mathcal{S} . Then, for each $S \in \mathcal{C}_{\text{IS}}$, $P, Q \in \mathcal{L}_{\mathcal{S}}$, and $b \in \mathbb{N}^+$, $\mathcal{S} \models \{b : P\} S \{0 : Q\}$ only if there exists an $S' \in \mathcal{C}_{\text{IS}}$ in which the repetition operator occurs at most once such that (a) $\mathcal{S} \models \{b : P\} S' \{0 : Q\}$ and (b) $\text{Th}(\mathcal{S}) \vdash \{b : P\} S' \{0 : Q\}$ implies $\text{Th}(\mathcal{S}) \vdash \{b : P\} S \{0 : Q\}$.*

Proof: Let $S \in \mathcal{C}_{\text{IS}}$ be such that the repetition operator occurs at least once in S . Then the following properties follow directly from the definitions involved ((1) and (2)) and the presented Hoare-like logic ((3) and (4)):

$$(1) \quad \mathcal{S} \models \{b : P\} S ; T \{0 : Q\} \text{ implies } \mathcal{S} \models \{b : P\} S \{0 : Q\};$$

- (2) $\mathcal{S} \models \{b : P\} S^\omega \{0 : Q\}$ implies $\mathcal{S} \models \{b : P\} S \{0 : Q\}$;
- (3) $\text{Th}(\mathcal{S}) \vdash \{b : P\} S \{0 : Q\}$ implies $\text{Th}(\mathcal{S}) \vdash \{b : P\} S ; T \{0 : Q\}$;
- (4) $\text{Th}(\mathcal{S}) \vdash \{b : P\} S \{0 : Q\}$ implies $\text{Th}(\mathcal{S}) \vdash \{b : P\} S^\omega \{0 : Q\}$.

Using these properties, the theorem is easily proved by induction on the number of occurrences of the repetition operator in S . \square

As a corollary of Theorem 1 we have that a completeness result for the set of all closed PGA terms of sort **IS** in which the repetition operator occurs at most once entails a completeness result for the set of all closed PGA terms of sort **IS**.

5 Example

In this section, we give an example of the use of the Hoare-like logic of asserted single-pass instruction sequences presented in Section 4. The example has only been chosen because it is simple and shows applications of most axioms and rules of inference of this Hoare-like logic (including R6 and R8).

For \mathcal{S} , we take an algebra of services that make up unbounded natural number counters. Each natural number counter service is able to process methods to increment the content of the counter by one (**incr**), to decrement the content of the counter by one (**decr**), and to test whether the content of the counter is zero (**iszero**). The derived service and service reply operations for these methods are as to be expected. $\Sigma_{\mathcal{S}}$ includes the sort **N** of natural numbers, the constant $0 : \rightarrow \mathbf{N}$, and the unary operators $\text{succ} : \mathbf{N} \rightarrow \mathbf{N}$, $\text{pred} : \mathbf{N} \rightarrow \mathbf{N}$, and $\text{nnc} : \mathbf{N} \rightarrow \mathbf{S}$. The interpretation of **N**, 0 , succ , and pred are as to be expected. The interpretation of nnc is the function that maps each natural number n to the service that makes up a counter whose content is n .

We claim that the closed PGA term $(-c.\text{iszero}; \#2; !; c.\text{decr})^\omega$ denotes an instruction sequence for setting the counter made up by service c to zero. That is, we claim $\{1 : \top\} (-c.\text{iszero}; \#2; !; c.\text{decr})^\omega \{0 : c = \text{nnc}(0)\}$. We prove this by means of the axioms and rules of inference given in Table 5.

It is sufficient to prove

- (1) $\{1 : c = \text{nnc}(0) \vee c = \text{nnc}(n+1)\} (-c.\text{iszero}; \#2; !; c.\text{decr})^\omega \{0 : c = \text{nnc}(0)\}$

because the claim follows from (1) by R8 and R10.

First, we prove $\{1 : c = nnc(0)\} -c.iszero ; \#2 ; ! ; c.decr \{0 : c = nnc(0)\}$:

- (2) $\{1 : c = nnc(0)\} -c.iszero \{2 : c = nnc(0)\}$
by A6;
- (3) $\{1 : c = nnc(0)\} -c.iszero ; \#2 \{1 : c = nnc(0)\}$
from (2) by A9 and R2;
- (4) $\{1 : c = nnc(0)\} -c.iszero ; \#2 ; ! \{0 : c = nnc(0)\}$
from (3) by A11 and R1;
- (5) $\{1 : c = nnc(0)\} -c.iszero ; \#2 ; ! ; c.decr \{0 : c = nnc(0)\}$
from (4) by A1 and R3.

Next, we prove $\{1 : c = nnc(n + 1)\} -c.iszero ; \#2 ; ! ; c.decr \{0 : c = nnc(n)\}$:

- (6) $\{1 : c = nnc(n + 1)\} -c.iszero \{1 : c = nnc(n + 1)\}$
by A6;
- (7) $\{1 : c = nnc(n + 1)\} -c.iszero ; \#2 \{2 : c = nnc(n + 1)\}$
from (6) by A9 and R1;
- (8) $\{1 : c = nnc(n + 1)\} -c.iszero ; \#2 ; ! \{1 : c = nnc(n + 1)\}$
from (7) by A11 and R2;
- (9) $\{1 : c = nnc(n + 1)\} -c.iszero ; \#2 ; ! ; c.decr \{0 : c = nnc(n)\}$
from (8) by A1, R10 and R1.

Assuming (1), we prove

- $\{1 : c = nnc(0) \vee c = nnc(n + 1)\}$
 $-c.iszero ; \#2 ; ! ; c.decr ; (-c.iszero ; \#2 ; ! ; c.decr)^\omega$
 $\{0 : c = nnc(0)\}$:
- (a) $\{1 : c = nnc(0)\}$
 $-c.iszero ; \#2 ; ! ; c.decr ; (-c.iszero ; \#2 ; ! ; c.decr)^\omega$
 $\{0 : c = nnc(0)\}$
from (5) by R3;
- (b) $\{1 : c = nnc(n + 1)\}$
 $-c.iszero ; \#2 ; ! ; c.decr ; (-c.iszero ; \#2 ; ! ; c.decr)^\omega$
 $\{0 : c = nnc(0)\}$
from (9) by R1;

- (c) $\{1 : c = nnc(0) \vee c = nnc(n + 1)\}$
 $-c.iszero ; \#2 ; ! ; c.decr ; (-c.iszero ; \#2 ; ! ; c.decr)^\omega$
 $\{0 : c = nnc(0)\}$
 from (a) and (b) by R6.

Because (c) has been derived assuming (1), (1) now follows by R5.

The example given above illustrates that proving instruction sequences correct can be quite tedious, even in a simple case. This can be largely attributed to the fact that instruction sequences do not need to be structured programs and not to the particular Hoare-like logic used. A verification condition generator and a proof assistant are anyhow indispensable when proving realistic instruction sequences correct.

6 Soundness and Completeness

This section is concerned with the soundness and completeness of the Hoare-like logic of asserted single-pass instruction sequences presented in Section 4. It was assumed in Section 4 that a signature $\Sigma_{\mathcal{S}}$ that includes specific sorts, constants and operators and a minimal $\Sigma_{\mathcal{S}}$ -algebra \mathcal{S} that satisfies specific conditions had been given. In this section, we intend to establish soundness and completeness for all algebras that could have been given. It is useful to introduce a name for these algebras: *service algebras*.

In this section, we write $\text{Th}(\mathcal{S})$, where \mathcal{S} is a service algebra, for the set of all formulas of $\mathcal{L}_{\mathcal{S}}$ that hold in \mathcal{S} .

The proof of the soundness theorem for the presented Hoare-like logic given below (Theorem 2) will make use of the following two lemmas. Recall that \vdash' stands for provability without applications of the repetition rule.

Lemma 1 *Let \mathcal{S} be a service algebra, and let $k, n \in \mathbb{N}^+$ be such that $k \leq n$. Then, for each $S, S' \in \mathcal{C}_{\mathbf{IS}}$, $P_1, \dots, P_n, Q_1, \dots, Q_n \in \mathcal{L}_{\mathcal{S}}$, and $b_1, \dots, b_n \in \mathbb{N}^+$, if $\{b_1 : P_1\} S^\omega \{0 : Q_1\}, \dots, \{b_n : P_n\} S^\omega \{0 : Q_n\} \vdash' \{b_k : P_k\} S ; S^\omega \{0 : Q_k\}$ then $\{b_1 : P_1\} S' \{0 : Q_1\}, \dots, \{b_n : P_n\} S' \{0 : Q_n\} \vdash' \{b_k : P_k\} S ; S' \{0 : Q_k\}$.*

Proof: This is easily proved by induction on the length of proofs, case distinction on the axiom applied in the basis step, and case distinction on the rule of inference last applied in the inductive step. \square

An important corollary of Lemma 1 is that, for all $i \in \mathbb{N}$ and $k \in \mathbb{N}^+$ with $k \leq n$, $\{b_1 : P_1\} S^\omega \{0 : Q_1\}, \dots, \{b_n : P_n\} S^\omega \{0 : Q_n\} \vdash' \{b_k : P_k\} S ; S^\omega \{0 : Q_k\}$

only if $\{b_1 : P_1\} S^i \{0 : Q_1\}, \dots, \{b_n : P_n\} S^i \{0 : Q_n\} \vdash' \{b_k : P_k\} S^{i+1} \{0 : Q_k\}$.

Lemma 2 *For each service algebra \mathcal{S} , set of asserted instruction sequences Ψ , and asserted instruction sequence ϕ , $\text{Th}(\mathcal{S}) \cup \Psi \vdash' \phi$ only if $\mathcal{S} \models \psi$ for all $\psi \in \Psi$ implies $\mathcal{S} \models \phi$.*

Proof: This is easily proved by induction on the length of proofs, case distinction on the axiom applied in the basis step, and case distinction on the rule of inference last applied in the inductive step. \square

Lemma 2 expresses that, if the repetition rule is dropped, the axioms and inference rules of the presented Hoare-like logic are strongly sound.

The following theorem is the soundness theorem for the presented Hoare-like logic.

Theorem 2 *For each service algebra \mathcal{S} and asserted instruction sequence ϕ , $\text{Th}(\mathcal{S}) \vdash \phi$ implies $\mathcal{S} \models \phi$.*

Proof: This is proved by induction on the length of proofs, case distinction on the axiom applied in the basis step, and case distinction on the rule of inference last applied in the inductive step. The only difficult case is the repetition rule (R5). We will only outline the proof for this case.

The following properties follow directly from the definition of $\mathcal{M}_{\mathcal{S}}$:

- (1) $\mathcal{M}_{\mathcal{S}} \models |S^0 ; \#0^b|_{b,0} \bullet t = \emptyset$;
- (2) $\mathcal{M}_{\mathcal{S}} \models |S^\omega|_{b,0} \bullet t = t'$ iff there exists an $j > 0$ such that:
 - for all $k \geq j$, $\mathcal{M}_{\mathcal{S}} \models |S^k ; \#0^b|_{b,0} \bullet t = t'$,
 - for all $k < j$, $\mathcal{M}_{\mathcal{S}} \models |S^k ; \#0^b|_{b,0} \bullet t = \emptyset$.

These properties could be largely proved in a formal way if the combined algebraic theory of $\mathcal{M}_{\mathcal{S}}$ developed in Sections 2 and 3 would be extended with projection operators and axioms for them as in [4].

The following properties follow directly from properties (1) and (2):

- (a) $\mathcal{S} \models \{b : P\} S^0 ; \#0^b \{0 : Q\}$;
- (b) $\mathcal{S} \models \{b : P\} S^\omega \{0 : Q\}$ iff, for all $i \geq 0$, $\mathcal{S} \models \{b : P\} S^i ; \#0^b \{0 : Q\}$.

Let $k, n \in \mathbb{N}^+$ be such that $k \leq n$, and let $S \in \mathcal{C}_{\mathbf{IS}}$, $P_1, \dots, P_n, Q_1, \dots, Q_n \in \mathcal{L}_{\mathbf{S}}$, and $b_1, \dots, b_n \in \mathbb{N}^+$. Then, from the hypotheses of R5 and Lemmas 1 and 2, it follows immediately that, for all $i \geq 0$, $\mathbf{S} \models \{b_1 : P_1\} S^i ; \#0^{b_1} \{0 : Q_1\}$ and \dots and $\mathbf{S} \models \{b_n : P_n\} S^i ; \#0^{b_n} \{0 : Q_n\}$ implies $\mathbf{S} \models \{b_k : P_k\} S^{i+1} ; \#0^{b_k} \{0 : Q_k\}$. From this and property (a), it follows by induction on i that, for all $i \geq 0$, $\mathbf{S} \models \{b_k : P_k\} S^i ; \#0^{b_k} \{0 : Q_k\}$. From this and property (b), it follows immediately that $\mathbf{S} \models \{b_k : P_k\} S^\omega \{0 : Q_k\}$. This completes the proof for the case of the repetition rule. \square

The line of the proof of Theorem 2 for the case that the rule of inference last applied is R5 is reminiscent of the line of the soundness proof in [9] for the case that the rule of inference last applied is the recursion rule for calls of recursive procedures. In the proof of Theorem 2, $S^i ; \#0^b$ is used instead of S^i to guarantee that b is never greater than the length of the approximations of S^ω .

There is a problem with establishing completeness for all service algebras. In the completeness proof, it has to be assumed that, for each service algebra \mathbf{S} , necessary intermediate conditions can be expressed in $\mathcal{L}_{\mathbf{S}}$. Therefore, completeness will only be established for all service algebras that are sufficiently expressive.

Let \mathbf{S} be a service algebra, and let $S \in \mathcal{C}_{\mathbf{IS}}$, $P, Q \in \mathcal{L}_{\mathbf{S}}$, $b \in \mathbb{N}^+$ and $e \in \mathbb{N}$. Then Q expresses the strongest post-condition of P and S for b and e on \mathbf{S} if $\mathbf{S} \models \{b : P\} S \{e : T\}$ and, for each state representing term t' for P , Q , and S with respect to \mathbf{S} , $\mathbf{S} \models Q[t']$ iff there exists a state representing term t for P , Q , and S with respect to \mathbf{S} such that $\mathbf{S} \models P[t]$ and $\mathcal{M}_{\mathbf{S}} \models |S|_{b,e} \bullet t = t'$.

Let \mathbf{S} be a service algebra. Then the language $\mathcal{L}_{\mathbf{S}}$ is expressive for $\mathcal{C}_{\mathbf{IS}}$ on \mathbf{S} if, for each $S \in \mathcal{C}_{\mathbf{IS}}$, $P \in \mathcal{L}_{\mathbf{S}}$, $b \in \mathbb{N}^+$, and $e \in \mathbb{N}$ with $\mathbf{S} \models \{b : P\} S \{e : T\}$, there exists a $Q \in \mathcal{L}_{\mathbf{S}}$ such that Q expresses the strongest post-condition of P and S for b and e on \mathbf{S} .

In the above definitions, $\mathbf{S} \models \{b : P\} S \{e : T\}$ is used to express that there exists a post-condition of P and S for b and e on \mathbf{S} .

The following remarks about the existence of strongest post-conditions may be useful for a clear understanding of the matter. For each $S \in \mathcal{C}_{\mathbf{IS}}$, $P \in \mathcal{L}_{\mathbf{S}}$, and $b \in \mathbb{N}^+$, one of the following is the case regarding the existence of a strongest post-condition:

- (1) there is no $e \in \mathbb{N}$ for which there exists a strongest post-condition of P and S for b and e ;

- (2) there is exactly one $e \in \mathbb{N}$ for which there exists a strongest post-condition of P and S for b and e and the strongest post-condition concerned is not equivalent to F ;
- (3) there is more than one $e \in \mathbb{N}$ for which there exists a strongest post-condition of P and S for b and e and the strongest post-condition concerned is equivalent to F .

We say that execution is convergent in S if it does not become inactive in S . Terminating in S is one way in which execution may be convergent in S , exiting S by going to the e th instruction following S is another way in which execution may be convergent in S , and exiting S by going to the e' th instruction following S , where $e' \neq e$, is still another way in which execution may be convergent in S . Now, (1) is the case if there is more than one way in which execution may be convergent in S , (2) is the case if there is exactly one way in which execution may be convergent in S , and (3) is the case if there is no way in which execution may be convergent in S .

The proof of the completeness theorem for the presented Hoare-like logic given below (Theorem 3) will make use of the following four lemmas.

Lemma 3 *Let \mathcal{S} be a service algebra. Then, for each $S \in \mathcal{C}_{\mathbf{IS}}$, $P, Q \in \mathcal{L}_{\mathcal{S}}$, $b \in \mathbb{N}^+$, and $e \in \mathbb{N}$, $\mathcal{S} \models \{b : P\} S^\omega \{e : Q\}$ only if $e = 0$.*

Proof: This is proved by distinguishing two cases: the repetition operator does not occur in S and the repetition operator occurs in S . The former case is easily proved by induction on $\text{len}(S)$. The latter case follows directly from the following corollary of the proof of Lemma 2.6 from [5]: for each $S \in \mathcal{C}_{\mathbf{IS}}$ in which the repetition operator occurs, there exists an S' in which the repetition operator does not occur such that $|S|_{b,e} = |S'^\omega|_{b,e}$. \square

Lemma 3 tells us that execution never exits an instruction sequence segment of the form S^ω .

The following lemma expresses that the axioms and rules of inference of the presented Hoare-like logic are complete for all instruction sequence segments of the form S^ω only if they are complete for all instruction sequence segments.

Lemma 4 *Let \mathcal{S} be a service algebra such that $\mathcal{L}_{\mathcal{S}}$ is expressive for $\mathcal{C}_{\mathbf{IS}}$ on \mathcal{S} . Assume that, for each $S \in \mathcal{C}_{\mathbf{IS}}$, $P, Q \in \mathcal{L}_{\mathcal{S}}$, $b \in \mathbb{N}^+$, and $e \in \mathbb{N}$, $\mathcal{S} \models \{b : P\} S^\omega \{e : Q\}$ implies $\text{Th}(\mathcal{S}) \vdash \{b : P\} S^\omega \{e : Q\}$. Then, for each $S \in \mathcal{C}_{\mathbf{IS}}$, $P, Q \in \mathcal{L}_{\mathcal{S}}$, $b \in \mathbb{N}^+$, and $e \in \mathbb{N}$, $\mathcal{S} \models \{b : P\} S \{e : Q\}$ implies $\text{Th}(\mathcal{S}) \vdash \{b : P\} S \{e : Q\}$.*

Proof: This is proved by induction on the structure of S . The cases that S is a single instructions follow, with the exception of the termination instruction after a case distinction, directly from one of the axioms (A1–A11) and the consequence rule (R10). The case that S is of the form S'^{ω} follows immediately from the assumption. What is left is the case that S is of the form $S_1 ; S_2$.

If $\mathcal{S} \models \{b:P\} S_1 ; S_2 \{e:Q\}$, then it follows from the definitions involved that:

- (1) if $b \leq \text{len}(S_1)$: for some $n > 0$, there exist $P_1, R_1, \dots, P_n, R_n \in \mathcal{L}_{\mathcal{S}}$ and $i_1, \dots, i_n \in \mathbb{N}^+$ such that $\mathcal{S} \models P \Rightarrow P_1 \vee \dots \vee P_n$ and, for each j with $1 \leq j \leq n$, R_j expresses the strongest post-condition of P_j and S_1 for b and i_j on \mathcal{S} and one of the following is the case:
 - (a) $1 \leq i_j \leq \text{len}(S_2)$,
 $\mathcal{S} \models \{b:P_j\} S_1 \{i_j:R_j\}$, and $\mathcal{S} \models \{i_j:R_j\} S_2 \{e:Q\}$;
 - (b) $i_j = \text{len}(S_2) + e$, $e > 0$,
 $\mathcal{S} \models \{b:P_j\} S_1 \{i_j:R_j\}$, and $\mathcal{S} \models R_j \Rightarrow Q$;
 - (c) $i_j = 0$, $e = 0$,
 $\mathcal{S} \models \{b:P_j\} S_1 \{i_j:R_j\}$, and $\mathcal{S} \models R_j \Rightarrow Q$;
- (2) if $b > \text{len}(S_1)$: $\mathcal{S} \models \{b - \text{len}(S_1):P\} S_2 \{e:Q\}$.

Case (1) is proved by distinguishing two subcases: the repetition operator does not occur in S_1 and the repetition operator occurs in S_1 . The former subcase is easily proved by induction on $\text{len}(S_1)$. The latter subcase follows directly from the above-mentioned corollary of the proof of Lemma 2.6 from [5] and Lemma 3. In either subcase, the existence of R_j 's that express the strongest post-conditions needed is guaranteed by the expressiveness property of $\mathcal{L}_{\mathcal{S}}$. Case (2) follows directly from the definitions involved.

In case (1), $\text{Th}(\mathcal{S}) \vdash \{b:P\} S_1 ; S_2 \{e:Q\}$ follows directly by the induction hypothesis, the first three concatenation rules (R1–R3), and the alternatives rule (R6). In case (2), it follows directly by the induction hypothesis and the last concatenation rule (R4). \square

The next lemma tells us that the axioms and inference rules of the presented Hoare-like logic is complete if provability can be identified with provability from a particular set of asserted single-pass instruction sequences; and the second next lemma expresses that the asserted single-pass instruction sequences concerned are provable.

Lemma 5 *Let \mathcal{S} be a service algebra such that $\mathcal{L}_{\mathcal{S}}$ is expressive for $\mathcal{C}_{\mathbf{IS}}$ on \mathcal{S} . For each $S \in \mathcal{C}_{\mathbf{IS}}$, let $x_1^S, \dots, x_{n_S}^S \in \mathcal{F}$ and $y_1^S, \dots, y_{n_S}^S \in \mathcal{F}$ be such that $\text{var}(S) = \{x_1^S, \dots, x_{n_S}^S\}$ and $\text{var}(S) \cap \{y_1^S, \dots, y_{n_S}^S\} = \emptyset$. For each $S \in \mathcal{C}_{\mathbf{IS}}$ and $b \in \mathbb{N}^+$, let P'_S be $x_1^S = y_1^S \wedge \dots \wedge x_{n_S}^S = y_{n_S}^S$, and let $Q'_{S,b} \in \mathcal{L}_{\mathcal{S}}$ be such that $Q'_{S,b}$ expresses the strongest post-condition of P'_S and S^ω for b and 0 on \mathcal{S} . For each $S \in \mathcal{C}_{\mathbf{IS}}$ and $b \in \mathbb{N}^+$, let $ub_{S,b} = \max\{b' \in \mathbb{N}^+ \mid b' = b \vee \#b' \text{ occurs in } S\}$. Then, for each $S' \in \mathcal{C}_{\mathbf{IS}}$, $P, Q \in \mathcal{L}_{\mathcal{S}}$, and $b \in \mathbb{N}^+$, $\mathcal{S} \models \{b : P\} S' \{0 : Q\}$ implies $\text{Th}(\mathcal{S}) \cup \{\{b' : P'_S\} S^\omega \{0 : Q'_{S,b'}\} \mid b' \leq ub_{S',b} \wedge S^\omega \text{ is a subterm of } S'\} \vdash \{b : P\} S' \{0 : Q\}$.*

Proof: This is proved by induction on the structure of S' . The cases that S' is a single instruction follow directly from one of the axioms (A2, A5, A8, A10, A11) and the consequence rule (R10). The case that S' is of the form $S_1 ; S_2$ is proved, using the induction hypothesis, in the same way as the case of concatenation in the proof of Lemma 4. What is left is the case that S' is of the form S^ω .

In the case that S' is of the form S^ω , it suffices to show that, for each $S \in \mathcal{C}_{\mathbf{IS}}$, $P, Q \in \mathcal{L}_{\mathcal{S}}$, and $b \in \mathbb{N}^+$, $\mathcal{S} \models \{b : P\} S^\omega \{0 : Q\}$ implies $\text{Th}(\mathcal{S}) \cup \{\{b : P'_S\} S^\omega \{0 : Q'_{S,b}\}\} \vdash \{b : P\} S^\omega \{0 : Q\}$.

Let $S \in \mathcal{C}_{\mathbf{IS}}$, $P, Q \in \mathcal{L}_{\mathcal{S}}$, and $b \in \mathbb{N}^+$, and let $z_1, \dots, z_{n_S} \in \mathcal{F}$ be such that $(\text{var}(S) \cup \text{var}(P) \cup \text{var}(Q) \cup \{y_1, \dots, y_{n_S}\}) \cap \{z_1, \dots, z_{n_S}\} = \emptyset$. Moreover, let P_1 be $P[z_1/y_1^S] \dots [z_{n_S}/y_{n_S}^S]$, let Q_1 be $Q[z_1/y_1^S] \dots [z_{n_S}/y_{n_S}^S]$, and let P_2 be $P_1[y_1^S/x_1^S] \dots [y_{n_S}^S/x_{n_S}^S]$. In the rest of this proof, a *state representing term* is a closed term of sort \mathbf{SF} that is a state representing term for P , Q , S , and $\{y_1^S, \dots, y_{n_S}^S\} \cup \{z_1, \dots, z_{n_S}\}$ with respect to \mathcal{S} . Assume $\mathcal{S} \models \{b : P\} S^\omega \{0 : Q\}$.

From $\{b : P'_S\} S^\omega \{0 : Q'_{S,b}\}$, it follows that $\{b : P'_S \wedge P_2\} S^\omega \{0 : Q'_{S,b} \wedge P_2\}$ (*) by the invariance rule (R7). We now show that $\mathcal{S} \models (Q'_{S,b} \wedge P_2) \Rightarrow Q_1$.

Let t' be a state representing term. Assume $\mathcal{S} \models (Q'_{S,b} \wedge P_2)[t']$. By the definition of $Q'_{S,b}$, there exists a state representing term t such that $\mathcal{S} \models P'_S[t]$ and $\mathcal{M}_{\mathcal{S}} \models |S^\omega|_{b,0} \bullet t = t'$ and $\mathcal{M}_{\mathcal{S}} \models |S|_{b,e'} \bullet t = \emptyset$ for all $e' \in \mathbb{N}$ with $e \neq e'$. Suppose $\mathcal{S} \models (\neg P_2)[t]$. From this, the just-mentioned properties of t , and the soundness of the invariance rule, it follows that $\mathcal{S} \models (\neg P_2)[t']$. This contradicts the assumption that $\mathcal{S} \models (Q'_S \wedge P_2)[t']$. Consequently, $\mathcal{S} \models P_2[t]$. From this, the first of the above-mentioned properties of t , and the fact that $\mathcal{S} \models (P'_S \wedge P_2) \Rightarrow P_1$, it follows that $\mathcal{S} \models P_1[t]$. From this, the assumption that $\mathcal{S} \models \{b : P\} S^\omega \{0 : Q\}$, and the soundness of the substitution rule (R9), it follows that $\mathcal{S} \models Q_1[t']$. This proves that $\mathcal{S} \models (Q'_{S,b} \wedge P_2) \Rightarrow Q_1$ (**).

From (*) and (**), it now follows by the consequence rule (R10) that

$\{b : P'_S \wedge P_2\} S^\omega \{0 : Q_1\}$. From this, it follows by the elimination rule (R8) that $\{b : \exists y_1^S, \dots, y_{n_S}^S \bullet (P'_S \wedge P_2)\} S^\omega \{0 : Q_1\}$. From this and the fact that $\mathcal{S} \models P_1 \Rightarrow \exists y_1^S, \dots, y_{n_S}^S \bullet (P'_S \wedge P_2)$, it follows that $\{b : P_1\} S^\omega \{0 : Q_1\}$ by the consequence rule. From this, it follows that $\{b : P\} S^\omega \{0 : Q\}$ by the substitution rule. \square

Lemma 6 *Let \mathcal{S} and, for each $S \in \mathcal{C}_{\text{IS}}$ and $b \in \mathbb{N}^+$, P'_S , and $Q'_{S,b}$ be as in Lemma 5. Then, for each $S \in \mathcal{C}_{\text{IS}}$ and $b \in \mathbb{N}^+$, $\text{Th}(\mathcal{S}) \vdash \{b : P'_S\} S^\omega \{0 : Q'_{S,b}\}$.*

Proof: Let $S \in \mathcal{C}_{\text{IS}}$ and $b \in \mathbb{N}^+$. Then, by the definition of $Q'_{S,b}$, $\mathcal{S} \models \{b : P'_S\} S^\omega \{0 : Q'_{S,b}\}$. From this, it follows that $\mathcal{S} \models \{b : P'_S\} S ; S^\omega \{0 : Q'_{S,b}\}$ because $|S^\omega|_{b,0} = |S ; S^\omega|_{b,0}$. From this and Lemma 5, it follows that $\text{Th}(\mathcal{S}) \cup \{\{b' : P'_S\} S^\omega \{0 : Q'_{S,b'}\} \mid b' \leq ub_{S;S^\omega,b}\} \vdash \{b : P'_S\} S ; S^\omega \{0 : Q'_{S,b}\}$, where $ub_{S,b}$ is defined as in Lemma 5. Because we have proved this for an arbitrary b , it follows by the repetition rule that $\text{Th}(\mathcal{S}) \vdash \{b : P'_S\} S^\omega \{0 : Q'_{S,b}\}$. \square

The lines of the proofs of Lemmas 5 and 6, which are mostly concerned with repetition, are reminiscent of the lines of the proofs of Lemmas 1 and 2 from [1], which are mostly concerned with calls of (parameterless) recursive procedures.

The following theorem is the completeness theorem for the presented Hoare-like logic. The weak form of completeness that can be proved is known as completeness in the sense of Cook because this notion of completeness originates from Cook [11].

Theorem 3 *For each service algebra \mathcal{S} such that $\mathcal{L}_{\mathcal{S}}$ is expressive for \mathcal{C}_{IS} on \mathcal{S} and each asserted instruction sequence ϕ , $\mathcal{S} \models \phi$ implies $\text{Th}(\mathcal{S}) \vdash \phi$.*

Proof: This result is an immediate consequence of Lemmas 3–6. \square

7 Concluding Remarks

We have presented a Hoare-like logic for proving the partial correctness of a single-pass instruction sequence as considered in program algebra and have shown that it is sound and complete in the sense of Cook. We have extended the asserted programs of Hoare logics with two natural numbers which represent conditions on how execution enters and exits an instruction sequence. By that we have prevented that pre- and post-conditions can be formulated in which aspects of input-output behaviour and flow of execution

are combined in ways that are unnecessary for proving (partial) correctness of instruction sequences. We believe that by the way in which we have extended the asserted programs of Hoare logics, the presented Hoare-like logic remains as close to Hoare logics as possible in the case where program segments with multiple entry points and multiple exit points have to be dealt with.

In contrast with most related work, we have neither taken ad hoc restrictions and features of machine- or assembly-level programs into account nor abstracted in an ad hoc way from instruction sequences as found in low-level programs. Moreover, unlike some related work, we have stuck to classical first-order logic for pre- and post-conditions. In particular, the separating conjunction and separating implication connectives from separation logics [20] are not used in pre- and post-conditions. Because of this, most related work, including the work reported upon in [17, 19, 21], is only loosely related.

Most closely related is the work reported upon in [24, 25]. The form of asserted instruction sequences is inspired by [25]. However, as explained in Section 1, their interpretation differs somewhat. Moreover, no attention is paid to soundness and completeness issues in [25]. An asserted program from [24] corresponds essentially to a set of asserted instruction sequences concerning the same instruction sequence fragment. The particular form of these asserted programs has the effect that proofs using the program logic from [24] involve a lot of auxiliary label manipulation.

References

- [1] K. R. Apt. Ten years of Hoare’s logic: A survey - Part I. *ACM Transactions on Programming Languages and Systems*, 3(4):431–483, 1981. doi:[10.1145/357146.357150](https://doi.org/10.1145/357146.357150).
- [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](https://doi.org/10.1016/S1567-8326(02)00018-8).
- [3] J. A. Bergstra and C. A. Middelburg. Program algebra with a jump-shift instruction. *Journal of Applied Logic*, 6(4):553–563, 2008. doi:[10.1016/j.jal.2008.07.001](https://doi.org/10.1016/j.jal.2008.07.001).

- [4] 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](https://doi.org/10.1007/s00236-012-0154-2).
- [5] 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](https://doi.org/10.2991/978-94-91216-65-7).
- [6] J. A. Bergstra and C. A. Middelburg. Instruction sequence based non-uniform complexity classes. *Scientific Annals of Computer Science*, 24(1):47–89, 2014. doi:[10.7561/SACS.2014.1.47](https://doi.org/10.7561/SACS.2014.1.47).
- [7] J. A. Bergstra and C. A. Middelburg. On instruction sets for Boolean registers in program algebra. *Scientific Annals of Computer Science*, 26(1):1–26, 2016. doi:[10.7561/SACS.2016.1.1](https://doi.org/10.7561/SACS.2016.1.1).
- [8] J. A. Bergstra and J. V. Tucker. Two theorems about the completeness of Hoare’s logic. *Information Processing Letters*, 15(4):143–149, 1982. doi:[10.1016/0020-0190\(82\)90095-3](https://doi.org/10.1016/0020-0190(82)90095-3).
- [9] E. M. Clarke. Programming language constructs for which it is impossible to obtain good Hoare axiom systems. *Journal of the ACM*, 26(1):129–147, 1979. doi:[10.1145/322108.322121](https://doi.org/10.1145/322108.322121).
- [10] M. Clint and C. A. R. Hoare. Program proving: Jumps and functions. *Acta Informatica*, 1(3):214–224, 1972. doi:[10.1007/BF00288686](https://doi.org/10.1007/BF00288686).
- [11] S. A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal of Computing*, 7(1):70–90, 1978. doi:[10.1137/0207005](https://doi.org/10.1137/0207005).
- [12] A. de Bruin. Goto statements: Semantics and deduction systems. *Acta Informatica*, 15(4):385–424, 1981. doi:[10.1007/BF00264536](https://doi.org/10.1007/BF00264536).
- [13] 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](https://doi.org/10.1007/978-3-642-69962-7).
- [14] R. W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science*, volume 19 of *Proceedings of Symposia in Applied Mathematics*, pages 19–32. American Mathematical Society, 1967.

- [15] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 583, 1969. doi:[10.1145/363235.363259](https://doi.org/10.1145/363235.363259).
- [16] W. A. Hodges. *Model Theory*, volume 42 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, Cambridge, 1993.
- [17] J. B. Jensen, N. Benton, and A. Kennedy. High-level separation logic for low-level code. In *POPL 2013*, pages 301–314. ACM Press, 2013. doi:[10.1145/2480359.2429105](https://doi.org/10.1145/2480359.2429105).
- [18] C. A. Middelburg. Instruction sequences as a theme in computer science. <https://instructionsequence.wordpress.com/>, 2015.
- [19] M. O. Myreen and M. J. C. Gordon. Hoare logic for realistically modelled machine code. In O. Grumberg and M. Huth, editors, *TACAS 2007*, volume 4424 of *Lecture Notes in Computer Science*, pages 568–582. Springer-Verlag, 2007. doi:[10.1007/978-3-540-71209-1_44](https://doi.org/10.1007/978-3-540-71209-1_44).
- [20] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS 2002*, pages 55–74. IEEE Computer Society Press, 2002.
- [21] A. Saabas and T. Uustalu. A compositional natural semantics and Hoare logic for low-level languages. *Theoretical Computer Science*, 373(3):273–302, 2007. doi:[10.1016/j.tcs.2006.12.020](https://doi.org/10.1016/j.tcs.2006.12.020).
- [22] 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](https://doi.org/10.1007/978-3-642-59851-7_2).
- [23] 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](https://doi.org/10.1007/978-3-642-17336-3).
- [24] G. Tan and A. W. Appel. A compositional logic for control flow. In E. A. Emerson and K. S. Namjoshi, editors, *VMCAI 2006*, volume 3855 of *Lecture Notes in Computer Science*, pages 80–94. Springer-Verlag, 2006. doi:[10.1007/11609773_6](https://doi.org/10.1007/11609773_6).

- [25] A. Wang. An axiomatic basis for proving total correctness of goto-programs. *BIT Numerical Mathematics*, 16(1):88–102, 1976. doi: [10.1007/BF01940782](https://doi.org/10.1007/BF01940782).
- [26] M. Wirsing. Algebraic specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 675–788. Elsevier, Amsterdam, 1990.