Acta Informatica 21, 293-320 (1984)

Acta Informatica

The Axiomatic Semantics of Programs Based on Hoare's Logic

J.A. Bergstra[1] and J.V. Tucker[2]

[1] Department of Computer Science, Centrum voor Wiskunde en Informatica, Kruislaan 413, NL-109855 Amsterdam, The Netherlands

[2] Department of Computer Studies, University of Leeds, Leeds LS29JT, UK

Summary. This paper is about the Floyd-Hoare Principle which says that the semantics of a programming language can be formally specified by axioms and rules of inference for proving the correctness of programs written in the language. We study the simple language WP of **while**-programs and Hoare's system for partial correctness and we *calculate* the relational semantics of WP as this is determined by Hoare's logic. This calculation is possible by using relational semantics to build a completeness theorem for the logic. The resulting semantics AX we call the *axiomatic relational semantics* for WP. This AX is *not* the conventional semantics for WP: it need not be effectively computable or deterministic, for example. A large number of elegant properties of AX are proved and the Floyd-Hoare Principle is reconsidered.

Introduction

Background

The idea that a programming language L can be defined by the axioms and rules of inference involved in proving properties of programs written in L originates in Floyd [19] and Hoare [22]. The beauty, and utility, of this *Floyd-Hoare Principle* are derived from the observation that, on taking a formal axiomatic system to be the ultimate source for the specification of language behaviour, the system provides

(1) formal criteria for the correctness of implementations of the language L; and

(2) a set of properties for each program S of L which may be formally verified and, if verified, apply in all implementations of the language L.

The application of the Floyd-Hoare Principle to the language L is commonly described as defining L by means of *axiomatic semantics.*

Among early and significant writings on the axiomatic method of language specification are Hoare [23], Hoare and Lauer [24], Lauer [27], Manna [29], Dijkstra [17, 18]. In particular, in Hoare and Wirth [25] there is an axiomatic semantics for a part of Pascal. Subsequently, axiomatic semantics have been invented for older languages, such as Algol 68 (Schwartz [36]), and have been used in the design of new languages, such as Euclid (London et al. [28]). Popular and useful introductory accounts of axiomatic semantics are included in McGettrick [31] and Pagan [34].

Objective

In this paper we present a thorough theoretical analysis of the Floyd-Hoare Principle in the original setting of Hoare [22]. We consider the simple language WP of **while**-programs and the axiomatic system now known as *Hoare's logic* which proves input-output specifications of the form $\{p\}S\{q\}$, where p and q are first-order statements and S is a program. Without any preconceived idea about the behaviour of programs, we carefully calculate the semantics of WP when all that is known about **while**-program computation is what can be proved in Hoare's logic. Such a calculation is possible using the general framework of *relational* semantics for programs and the following idea about completeness theorems in logic:

Completeness Principle. *Let L be a formal logical system analysing a property P. A completeness theorem for L with respect to a formal semantics S for P is a statement confirming that semantics S characterises the property P, as P is determined by L.*

Thus, we will construct a relational semantics AX for WP based on Hoare's logic and prove that Hoare's logic is complete with respect to partial correctness under the semantics AX; this confirms that AX characterises the relational meaning of WP as far as Hoare's logic is concerned. Then we will study AX in detail and catalogue some of its intriguing properties; for example, AX is not a conventional semantics of WP for it is not computable and not deterministic! In consequence, the Floyd-Hoare Principle does not accomplish the task of defining this programming language in the natural way, as offered by first-order Hoare's logic. We consider the Floyd-Hoare Principle in the wake of our results in Sections 9 and 10.

Overview

It will be helpful if we prolong this introduction by summarising what we do in the paper. First, here are some words about Hoare's logic:

Hoare's logic consists of axioms and rules for manipulating specified programs $\{p\}S\{q\}$; and a first-order axiomatic specification (Σ, T) for the data types on which the programs compute. Each structure A of signature Σ that is

a model of (Σ, T) represents an implementation of (Σ, T). The data type specification (Σ, T) and the specified programs $\{p\}\, S\{q\}$ cooperate in the logic via the Rule of Consequence which is applied to first-order statements provable from T. The set of all specified programs provable in Hoare's logic we denote $HL(\Sigma, T)$. This essential material, including many derived rules and mathematical results for the system, is documented in Sections 1 and 2.

In Section 3, we define a relational semantics $AX(\Sigma, T)$ for WP based on $HL(\Sigma, T)$ and we refer to it as the *axiomatic relational semantics of WP*. This axiomatic semantics determines a new partial correctness semantics for specified programs. In Section 4, we prove:

Completeness Theorem. *Let (Σ, T) be any first-order data type specification. Let $\{p\}\, S\{q\}$ be any first-order specified program. Then*

$$HL(\Sigma, T) \vdash \{p\}\, S\{q\} \text{ if, and only if, } \quad AX(\Sigma, T) \models \{p\}\, S\{q\}.$$

The proof of this theorem is relatively complicated and we give it in detail because *the semantical definition of $AX(\Sigma, T)$ and the demonstration of completeness may be generalised to any logical system for partial correctness for any programming language, provided the system satisfies the generalised derived rules and metamathematical statements involved in the proof.*

Such a completeness theorem is impossible using the standard partial correctness semantics based on, say, the operational semantics of WP: see [10].

We study computation on a single structure A using the axiomatic semantics $AX(A)$ of the Hoare's logic $HL(A)$ made by allowing the set $Th(A)$ of all true first-order statements about A as data type specification. Of especial interest is the relationship between $AX(A)$ and the standard semantics of WP on A. Among the many results of Sections 5 and 6 are:

Theorem. *The operational semantics $OP(A)$ of WP on a minimal structure A is faithfully embedded in the axiomatic semantics $AX(A)$.*

If the first-order assertion language is expressive for WP with respect to its operational semantics $OP(A)$ on A then $OP(A) = AX(A)$. In particular, for the standard model of arithmetic $\mathbb{N}$, $OP(\mathbb{N}) = AX(\mathbb{N})$.

Theorem. *For any program S of WP the operational semantics $OP(A)(S)$ of S on A is recursively enumerable in $Th(A)$ while the axiomatic semantics $AX(A)(S)$ of S on A is co-recursively enumerable in $Th(A)$.*

There is a program S of WP on Presburger Arithmetic P such that $OP(P)(S) \subsetneqq AX(P)(S)$.

In Section 7, we consider the completeness of $HL(A)$ with respect to $AX(A)$ in greater detail. And in Section 8 we construct a structure A and a program S such that $AX(A)(S)$ is nondeterministic. We conclude the paper with further discussion of the project in Section 9 and Section 10.

Prerequisites

Our interest in axiomatic semantics began with our [6] written in collaboration with J. Tiuryn (see also Meyer and Halpern [32, 33]). However, this paper

owes more to our series of articles on the role of the data type specification in Hoare's logic [7–13]. There we found a paucity of significant completeness theorems for the logic which is the clue to the present project: it is a consequence of the Completeness Principle that *if a logical system is not complete for a semantics then the system is not talking about that semantics.* These issues are discussed in detail in [10]. In addition to Hoare [22], the reader must be familiar with the basic results of Cook [14], for which Apt [1] may be consulted. To master all the arguments that follow acquaintence with the entire series [7–13] is recommended, and may be necessary.

1. Specifications and Programs

1.1. Syntax

The first-order language $L(\Sigma)$ of some signature Σ is based upon a set Var of variables $x_1, x_2, \ldots$ and its constant, function and relational symbols are those of Σ, together with the equality relation. We assume $L(\Sigma)$ possesses the usual logical connectives and quantifiers; and the set of algebraic expressions of terms over Σ we denote $T(\Sigma)$.

If T is a set of assertions of $L(\Sigma)$ and $p \in L(\Sigma)$ is formally provable from T then we write $T \vdash p$. Such a set T of formulae is usually called a theory over Σ, but more appropriate for our purposes is to call the pair (Σ, T) a *first-order data type specification.*

Using the syntax of $L(\Sigma)$, the set $WP(\Sigma)$ of all **while**-programs over Σ is defined in the customary way.

By a *specified* or *asserted program* we mean a triple of the form $\{p\}\, S\, \{q\}$ where $S \in WP(\Sigma)$ and $p, q \in L(\Sigma)$.

Thus, here first-order languages are used as program specification languages and data type specification languages.

1.2. Specification Semantics

The semantics of $L(\Sigma)$ is the standard satisfaction semantics of model theory. A *state* over a structure A is a map

$$\sigma\colon \mathrm{Var} \to A$$

assigning a value $\sigma(x)$ in A to each and every variable x of $L(\Sigma)$; let State(A) denote the set of all states over A. The validity of $p \in L(\Sigma)$ at a state σ we write $A \models p(\sigma)$; and if $A \models p(\sigma)$ for every state $\sigma \in \mathrm{State}(A)$ then we write $A \models p$. The set of all sentences of $L(\Sigma)$ which are valid in A is called the first-order theory of A and is denoted $Th(A)$.

The class of all models of a specification (Σ, T) is denoted $\mathrm{Mod}(\Sigma, T)$. If $p \in L(\Sigma)$ and $A \models p$ for each $A \in \mathrm{Mod}(\Sigma, T)$ then we write $\mathrm{Mod}(\Sigma, T) \models p$. According to our Completeness Principle in the Introduction the semantics of the first-order data type specification (Σ, T) is $\mathrm{Mod}(\Sigma, T)$:

1.3. Gödel Completeness Theorem. *Let (Σ, T) be a specification. For each $p \in L(\Sigma)$,*

$$T \vdash p \quad \textit{if, and only if,} \quad \mathrm{Mod}(\Sigma, T) \models p.$$

Of course, the Completeness Principle was first understood in the contrasting of Theorem 1.3 with the Gödel Incompleteness Theorem.

1.4. Program Semantics

Since the purpose of this paper is to investigate a new and non-standard semantics for $WP(\Sigma)$ it is wise to comment on the familiar and standard semantics for $WP(\Sigma)$. For the semantics of $WP(\Sigma)$ on a structure A the reader is free to choose any "conventional" account of **while**-program computation to make comparisons with the new axiomatic semantics. Informally, of course, we expect all semantics for $WP(\Sigma)$ defined elsewhere, in normal circumstances, to be essentially equivalent.

However, since there is, as yet, no single framework for studying semantics in an unified way, there is no formal and general criterion for the equivalence or isomorphism of any two semantic definitions of program behaviour. In consequence, there is no way of refining the idea of a "conventional" semantics for **while**-programs by means of a formal definition that identifies the standard semantics uniquely up to isomorphism. A notable attempt at the problem of *rigorously* comparing the disparate methods of defining the semantics of **while**-programs is made in Greif and Meyer [21].

Among the existing treatments of **while**-programs, our preference is the operational semantics defined in deBakker [5], generalised from the natural numbers $\mathbf{N}$ to an abstract structure A. The meaning of $S \in WP(\Sigma)$ on A is defined, by induction on the structure of S, to be a *state transformation map*

$$O_A(S)\colon \mathrm{State}(A) \to \mathrm{State}(A).$$

The map $O_A(S)$ is a partial function and we write $O_A(S)(\sigma)\downarrow\tau$, if $O_A(S)(\sigma)$ is defined and has the value τ, and $O_A(S)(\sigma)\uparrow$ if $O_A(S)(\sigma)$ is not defined. Let us note that if S has n variables then, for the purpose of analysing S, we may faithfully represent $\mathrm{State}(A)$ by A^n and faithfully represent $O_A(S)$ by a map

$$\hat{O}_A(S)\colon A^n \to A^n$$

in an obvious way.

Although the function $O_A(S)$ reflects intuition, later on, we will replace it by a relation $OP(A)(S)$; $OP(A)(S)$ is just the graph of the function $O_A(S)$.

1.5. Partial Correctness

Putting together the semantics of $L(\Sigma)$ and $WP(\Sigma)$, we define the *(standard) partial correctness semantics* of the specified programs: the specified program $\{p\}S\{q\}$ is *valid* on A, written $OP(A)\models\{p\}S\{q\}$, if for each initial state $\sigma\in\text{State}(A)$, $A\models p(\sigma)$ implies either $O_A(S)(\sigma)\downarrow\tau$ and $A\models p(\tau)$ or $O_A(S)(\sigma)\uparrow$. In particular, if $OP(A)\models\{p\}S\{q\}$ then we say that S is *partially correct with respect to the precondition p and postcondition q under its standard operational semantics.* The specified program $\{p\}S\{q\}$ is *valid* for the data type specification (Σ, T) if $OP(A)\models\{p\}S\{q\}$ for each $A\in\text{Mod}(\Sigma, T)$.

We define the *standard partial correctness theory of a structure A* as the set

$$SPC(A)=\{\{p\}S\{q\}:OP(A)\models\{p\}S\{q\}\}$$

and the *standard partial correctness theory of a specification* (Σ, T) as the set

$$SPC(\Sigma, T)=\{\{p\}S\{q\}:\quad \text{for all}\quad A\in\text{Mod}(\Sigma, T)\quad OP(A)\models\{p\}S\{q\}\}$$

and in general for a class K of Σ-structures:

$$SPC[K]=\{\{p\}S\{q\}:\quad \text{for all}\quad A\in K\quad OP(A)\models\{p\}S\{q\}\}$$

The following identity connects both notations:

$$SPC(\Sigma, T)=SPC[\text{Mod}(\Sigma, T)]$$

1.6. Computability

Foremost among the properties common to the several "conventional" semantics for **while**-programs is the property that **while**-programs can compute all and only partial recursive functions on the standard model of arithmetic

$$\mathbb{N}=(\{0, 1, \ldots\}: 0, x+1,\ x+y,\ x\times y).$$

In particular, with reference to our operational semantics, we note that the representation

$$\hat{O}_{\mathbb{N}}(S):\ \mathbb{N}^n\to\mathbb{N}^n$$

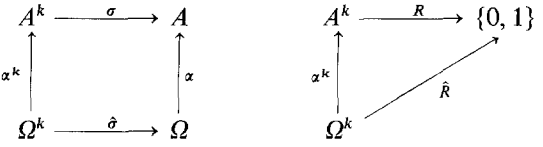
is partial recursive for every $S\in WP(\Sigma_{\mathbb{N}})$. These remarks are true of the simpler structure Presburger Arithmetic

$$P=(\{0, 1, \ldots\}: 0, x+1).$$

Later, in Section 6, we will contrast these familiar results with theorems about the effective computability of the axiomatic semantics and thereby show that the semantics is indeed nonstandard. We consider computability issues in a general setting using the ideas of Rabin [35] and Mal'cev [30].

A structure A has an *effective enumeration* when there is a recursive set Ω of natural numbers and a surjection α: $\Omega\to A$ such that for each k-ary oper-

ation σ and k-ary relation R of A there exist recursive functions $\hat{\sigma}$ and $\hat{R}$ which commute the following diagrams:



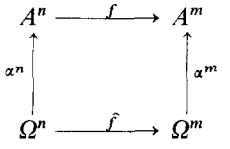
wherein $\alpha^k(n_1, \ldots, n_k) = (\alpha n_1, \ldots, \alpha n_k)$ and R is identified with its characteristic function.

A structure A is *computable* when there is an effective enumeration α such that the relation $\equiv_\alpha$ defined by

$$n \equiv_\alpha m \quad \text{if, and only if,} \quad \alpha n = \alpha m \text{ in } A$$

is recursive.

Let A be a structure computable under enumeration α. Then the partial function $f\colon A^n \to A^m$ is *computable under* α if there is a partial recursive function $\hat{f}\colon \Omega^n \to \Omega^m$ that commutes the following diagram:



With these concepts we may formulate a partial generalisation of our opening remarks.

1.7. Lemma. *Let A be a computable structure with signature Σ. Then for every $S \in WP(\Sigma)$ the partial function*

$$\hat{O}_A(S)\colon A^n \to A^n$$

is computable.

Thus, **while**-programs equipped with their standard semantics define computable partial functions on a computable structure. However, in general, not every computable function on a computable structure is definable by a **while**-program. This is because one can computably search all of a computable structure A via its enumeration, but the possibility that a **while**-program can search A depends on the constants named in the signature of A. A *minimal structure* is a structure finitely generated by elements named as constants in its signature.

2. Hoare's Logic

Hoare's logic for $WP(\Sigma)$ with data type specification (Σ, T) and assertion language $L(\Sigma)$ has the following axioms and proof rules for proving specified programs: Let $S, S_1, S_2 \in WP(\Sigma)$; $p, q, p_1, q_1, r \in L(\Sigma)$; $b \in L(\Sigma)$, a quantifier-free formula.

Assignment axiom scheme: For $e \in T(\Sigma)$ and $x \in \mathrm{Var}$ and $p \in L(\Sigma)$ the specified program

$$\{p[e/x]\}\ x := e\{p\}$$

is an axiom, where $p[e/x]$ stands for the result of substituting e for free occurrences of x in p.

Composition rule:

$$\frac{\{p\}\ S_1\{r\},\ \{r\}\ S_2\{q\}}{\{p\}\ S_1; S_2\{q\}}$$

Conditional rule:

$$\frac{\{p \wedge b\}\ S_1\{q\},\ \{p \wedge \neg b\}\ S_2\{q\}}{\{p\}\ \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \textbf{fi}\ \{q\}}$$

Iteration rule:

$$\frac{\{p \wedge b\}\ S\{p\}}{\{p\}\ \textbf{while}\ b\ \textbf{do}\ S\ \textbf{od}\ \{p \wedge \neg b\}}$$

Consequence rule:

$$\frac{p \to p_1,\ \{p_1\}\ S\{q_1\},\ q_1 \to q}{\{p\}\ S\{q\}}$$

Specification axiom scheme: Each $p \in L(\Sigma)$ such that $T \vdash p$ is an axiom.

The set of specified programs derivable from these axioms by the proof rules we denote $HL(\Sigma, T)$ and we write $HL(\Sigma, T) \vdash \{p\}\ S\{q\}$ in place of $\{p\}\ S\{q\} \in HL(\Sigma, T)$.

In the thorough examination of this system that follows we will employ a number of simple derived rules and proof-theoretic results; most of these we have established in our series [7–13] and are recalled here; first the derived rules:

2.1. Disjunction rule

$$\frac{\{p\}\ S\{q\},\ \{r\}\ S\{t\}}{\{p \vee r\}\ S\{q \vee t\}}$$

2.2. Conjunction rule

$$\frac{\{p\}\ S\{q\},\ \{r\}\ S\{t\}}{\{p \wedge r\}\ S\{q \wedge t\}}$$

2.3. Invariant rule. *If S and p have no free variables in common then* $HL(\Sigma, T) \vdash \{p\}\ S\{p\}$

2.4. $\forall$-Rule. *If x is not a free variable of S or p then*

$$\frac{\{p\}\ S\{q\}}{\{p\}\ S\{\forall x q\}}$$

2.5. $\exists$-Rule. *If x is not a free variable of S or q then*

$$\frac{\{p\}\ S\{q\}}{\{\exists x p\}\ S\{q\}}$$

Next we record the following metamathematical results taken from [8, 9].

2.6. Finiteness Lemma. *If* $HL(\Sigma, T) \vdash \{p\}\, S\, \{q\}$ *then for some closed* $\phi \in L(\Sigma)$ *we have*

$$T \vdash \phi \quad \text{and} \quad HL(\Sigma, \{\phi\}) \vdash \{p\}\, S\, \{q\}.$$

2.7. Deduction Lemma. *Let* $\phi \in L(\Sigma)$ *be closed and suppose that*

$$HL(\Sigma, T \cup \{\phi\}) \vdash \{p\}\, S\, \{q\}$$

then

$$HL(\Sigma, T) \vdash \{\phi \wedge p\}\, S\, \{q\}$$

2.8. Constants Lemma. *Let* $\Sigma_c = \Sigma \cup \{c\}$ *where* c *is a new constant symbol and suppose that*

$$HL(\Sigma_c, T) \vdash \{p\}\, S\, \{q\}$$

where $T \subset L(\Sigma)$ *and* $p, q \in L(\Sigma_c)$. *Then for any variable* x *not in* S, p *or* q

$$HL(\Sigma, T) \vdash \{p[x/c]\}\, S\, \{q[x/c]\}.$$

In addition, we recall the basic results of Cook [14] about the soundness and completeness of Hoare's logic with respect to its standard semantics:

2.9. Soundness Theorem. $HL(\Sigma, T) \subset SPC(\Sigma, T)$.

A specification (Σ, T) is *complete* if for any sentence $p \in L(\Sigma)$ either $T \vdash p$ or $T \vdash \neg p$. For any structure A, $Th(A)$ is complete.

The assertion language $L(\Sigma)$ is *expressive* for $WP(\Sigma)$ on Σ-structure A if for every $p \in L(\Sigma)$ and every $S \in WP(\Sigma)$ the strongest postcondition

$$sp_A(p, S) = \{\sigma \in \mathrm{State}(A) : \exists \tau [O_A(S)(\tau) \downarrow \sigma \,\&\, A \models p(\tau)]\}$$

is definable by a formula of $L(\Sigma)$.

2.10. Cook's completeness Theorem. *Let* (Σ, T) *be a complete specification and let* $A \in \mathrm{Mod}(\Sigma, T)$. *If* $L(\Sigma)$ *is expressive for* $WP(\Sigma)$ *on* A *then*

$$HL(\Sigma, T) = SPC(A)$$

Observe that $HL(A) = HL(\Sigma, Th(A))$ is the strongest Hoare logic for analysing program correctness on A because it is equipped with all first-order facts about A; and $HL(A)$ is complete whenever $L(\Sigma)$ is expressive.

2.11. Corollary. *For the standard model of arithmetic* $\mathbb{N}$

$$HL(\mathbb{N}) = PC(\mathbb{N})$$

We conclude with a fact from [7] we will use in Section 5.

2.12. Lemma. *Let* A *be any structure of signature* Σ *and let* ϕ *and* ψ *be any state formulae of the form*

$$\phi(x) \equiv \bigwedge_{i=1}^{n} x_i = t_i \quad \text{and} \quad \psi(x) \equiv \bigwedge_{i=1}^{n} x_i = r_i$$

where $x_i \in \mathrm{Var}$ *and* $t_i, r_i \in T(\Sigma)$. *Let* $S \in WP(\Sigma)$ *have the same variables* $x = (x_1, \ldots, x_n)$. *If* $A \models \phi(\sigma)$ *implies* $OP(A)(S)(\sigma)\downarrow$, *and* $OP(A) \models \{\phi\}\, S\, \{\psi\}$, *then* $HL(A) \vdash \{\phi\}\, S\, \{\psi\}$.

3. Axiomatic Semantics

Using Hoare's logic $HL(\Sigma, T)$ for the data type specification (Σ, T) we will define a semantics $AX(\Sigma, T)$ for $WP(\Sigma)$ over the class $\mathrm{Mod}(\Sigma, T)$ of implementations of (Σ, T). Before defining this axiomatic semantics we describe a general scheme for formulating a relational semantics over any class of interpretations.

3.1. *Program Semantics for a Class*

Let K be a class of structures of signature Σ. A *relational semantics* M for the set $WP(\Sigma)$ over K is a family

$$M = M[K] = \{M_A \colon A \in K\}$$

wherein each M_A is a mapping which assigns to each $S \in WP(\Sigma)$ a relation

$$M_A(S) \subset \mathrm{State}(A) \times \mathrm{State}(A).$$

Often we will prefer to write $M(A)$ rather than M_A. Square brackets will always contain a class description, whereas subscripts A denote a particular structure.

3.2. *Partial Correctness*

Let $M = \{M_A \colon A \in K\}$ be a relational semantics over a class K. A specified program $\{p\}\, S\, \{q\}$ is *partially correct on* A *with respect to* M_A if for each $(\sigma, \tau) \in M_A(S)$, $A \models p(\sigma)$ implies $A \models q(\tau)$; and we write $M_A \models \{p\}\, S\, \{q\}$ in such circumstances. Furthermore, the specified program $\{p\}\, S\, \{q\}$ is *partially correct over* K *with respect to* M if it is partially correct on A with respect to M_A for each $A \in K$; and we write $M \models \{p\}\, S\, \{q\}$ in such circumstances. Thus:

$$M \models \{p\}\, S\, \{q\} \quad \text{if, and only if, for each } A \in K, \quad M_A \models \{p\}\, S\, \{q\}.$$

3.3 *Correctness Theories*

The *partial correctness theory* of $WP(\Sigma)$ on A with respect to M_A is the set

$$PC(M_A) = \{\{p\}\, S\, \{q\} \colon M_A \models \{p\}\, S\, \{q\}\}$$

The partial correctness theory of $WP(\Sigma)$ over K with respect to M is the set

$$PC(M) = \{\{p\}\, S\, \{q\} \colon M \models \{p\}\, S\, \{q\}\}$$

Clearly

$$PC(M) = \bigcap_{A \in K} PC(M_A)$$

Note that the following implication holds:

$$\text{for all } S \in WP(\Sigma),\ M_A(S) \subseteq M'_A(S) \quad \text{implies} \quad PC(M'_A) \subseteq PC(M_A).$$

3.4. *Operational Semantics*

Recalling 1.4, we (re-)define the operational semantics of $WP(\Sigma)$ over a given class K to be the family

$$OP[K] = \{OP(A) \colon A \in K\}$$

wherein for $S \in WP(\Sigma)$

$$(\sigma, \tau) \in OP(A)(S) \quad \text{if, and only if,} \quad O_A(S)(\sigma) \downarrow \tau.$$

Note that $PC(OP[K]) = SPC[K]$ and $PC(OP(A)) = SPC(A)$.

3.5. *Axiomatic Semantics*

Let (Σ, T) be a data type specification. The axiomatic semantics of $WP(\Sigma)$ over $K = \mathrm{Mod}(\Sigma, T)$ is a relational semantics determined by $HL(\Sigma, T)$. This semantics corresponds to the family

$$AX(\Sigma, T) = AX(\Sigma, T)[\mathrm{Mod}(\Sigma, T)] = \{AX(\Sigma, T)_A \colon A \in \mathrm{Mod}(\Sigma, T)\}$$

wherein the meaning function $AX(\Sigma, T)_A$ assigns to each $S \in WP(\Sigma)$ the relation

$$AX(\Sigma, T)_A(S) = \{(\sigma, \tau) \in \mathrm{State}(A) \times \mathrm{State}(A) \colon \text{for any } p, q \in L(\Sigma), \text{ if } HL(\Sigma, T) \vdash \{p\}\, S\, \{q\} \text{ then } A \models p(\sigma) \text{ implies } A \models q(\tau)\}.$$

Thus, informally, we say that $(\sigma, \tau) \in AX(\Sigma, T)_A(S)$ *if and only if the input state* σ *and output state* τ *are consistent with every provably correct specification of the program* S *given the data type axiomatisation* (Σ, T) *for* A.

Applying the notations of Definitions 3.2 and 3.3 we write:

$$AX(\Sigma, T) \models \{p\}\, S\, \{q\} \text{ if, and only if, for each } A \in K,\ AX(\Sigma, T)_A \models \{p\}\, S\, \{q\};$$

and

$$PC(AX(\Sigma, T)) = \cap \{PC(AX(\Sigma, T)_A) \colon A \in \mathrm{Mod}(\Sigma, T)\}.$$

For the remainder of this section we examine the axiomatic semantics on a single interpretation A.

To compute on the structure A using **while**-programs under axiomatic semantics it is necessary to choose a data type specification (Σ, T) for A and then apply $AX(\Sigma, T)_A$ as described in Definition 3.5. It *appears* that the axiomatic semantics of $WP(\Sigma)$ on A is not uniquely determined by A (as is the

case for operational semantics, of course), but that the axiomatic semantics depends on A and (Σ, T): *Can different data type specifications allow different relations to be computed*? We show that the answer to this question is: No.

Consider the complete first-order specification of the structure A, namely the full first-order theory $Th(A)$ of A. Recall from Section 2,

$$HL(A) = HL(\Sigma, Th(A))$$

which maximises the set of provable information about **while**-program behaviour obtainable from Hoare's proof system.

3.6. Definition. The *full axiomatic semantics* of $WP(\Sigma)$ on A is $AX(A) = AX(\Sigma, Th(A))_A$. Thus, for $S \in WP(\Sigma)$,

$$AX(A)(S) = \{(\sigma, \tau): \text{for all } p, q \in L(\Sigma) \text{ if } HL(A) \vdash \{p\}\, S\, \{q\} \text{ then } A \models p(\sigma) \text{ implies } A \models q(\sigma)\}.$$

Furthermore for a class K we have the relational semantics $AX = AX[K]$ defined by $AX[K] = \{AX(A): A \in K\}$. (As a notational consequence we find: $AX[K](A) = AX(A) = AX[\{A\}]$ for $A \in K$.)

3.7. Uniqueness Theorem. *Let (Σ, T) be a specification. For each $S \in WP(\Sigma)$ and $A \in \mathrm{Mod}(\Sigma, T)$ it is the case that*

$$AX(\Sigma, T)_A(S) = AX(A)(S).$$

Proof. First, we show that $AX(A)(S) \subset AX(\Sigma, T)_A(S)$; this is straight-forward. Let $(\sigma, \tau) \in AX(A)(S)$ and suppose that for some $p, q \in L(\Sigma)$, $HL(\Sigma, T) \vdash \{p\}\, S\, \{q\}$. Then, trivially, $HL(A) \vdash \{p\}\, S\, \{q\}$. Since $(\sigma, \tau) \in AX(A)(S)$ by assumption, we know that $A \models p(\sigma)$ implies $A \models q(\tau)$. We have deduced that $(\sigma, \tau) \in AX(\Sigma, T)(S)$.

Consider now the converse: let $(\sigma, \tau) \in AX(\Sigma, T)(S)$ and suppose that for some $p, q \in L(\Sigma)$, $HL(A) \vdash \{p\}\, S\, \{q\}$; we must prove that $A \models p(\sigma)$ implies $A \models q(\tau)$.

By the Finiteness Lemma 2.6, there is a closed first-order assertion $\phi \in L(\Sigma)$ such that $A \models \phi$ and

$$HL(\Sigma, T \cup \{\phi\}) \vdash \{p\}\, S\, \{q\}.$$

By the Deduction Lemma 2.7,

$$HL(\Sigma, T) \vdash \{p \wedge \phi\}\, S\, \{q\}$$

Since $(\sigma, \tau) \in AX(\Sigma, T)_A(S)$ we may deduce that

$$A \models (p \wedge \phi)(\sigma) \quad \text{implies} \quad A \models q(\tau)$$

and because $A \models \phi$ we know that

$$A \models p(\sigma) \quad \text{implies} \quad A \models q(\tau). \qquad \square$$

Thus, *the axiomatic semantics of $WP(\Sigma)$ on a structure A defined by $HL(\Sigma, T)$ is independent of the data type specification (Σ, T) for A.* And the importance of $AX(A)$ is assured by the following observation:

3.8. Corollary. *Let* (Σ, T) *be a specification. Let* $S \in WP(\Sigma)$ *and* $p, q \in L(\Sigma)$. *For each* $A \in \mathrm{Mod}(\Sigma, T)$

$$AX(\Sigma, T)_A \models \{p\}\, S\, \{q\} \quad \textit{if, and only if,} \quad AX(A) \models \{p\}\, S\, \{q\}.$$

Furthermore:

$$AX(\Sigma, T) \models \{p\}\, S\, \{q\} \quad \textit{if, and only if, for each} \quad A \in \mathrm{Mod}(\Sigma, T),\ AX(A) \models \{p\}\, S\, \{q\}.$$

Finally, we record a technical fact of use in the next section.

Let B be an expansion of a structure A of signature Σ, namely B is A augmented with new constants, operations and relations and so $A = B|_\Sigma$.

3.9. Lemma. *If* B *is an expansion of* A *then for every* $S \in WP(\Sigma)$

$$AX(B)(S) \subset AX(A)(S)$$

and this entails that

$$AX(A) \models \{p\}\, S\, \{q\} \quad \textit{implies} \quad AX(B) \models \{p\}\, S\, \{q\}$$

for every $p, q \in L(\Sigma)$.

4. Soundness and Completeness of Hoare's Logic

We will now prove that Hoare's logic is sound and complete with respect to the axiomatic semantics of Definition 3.5:

4.1. Theorem. *Let* (Σ, T) *be a data type specification. For each program* $S \in WP(\Sigma)$ *and any assertions* $p, q \in L(\Sigma)$,

$$HL(\Sigma, T) \vdash \{p\}\, S\, \{q\} \quad \textit{if, and only if,} \quad AX(\Sigma, T) \models \{p\}\, S\, \{q\}.$$

Using the Completeness Principle, we interpret the theorem as a statement that *confirms* that the formulae of Definition 3.5 define the semantics of **while**-programs according to Hoare's logic. The proof of the soundness of $HL(\Sigma, T)$ with respect to $AX(\Sigma, T)$ is trivial, but completeness requires a longish proof which uses some standard techniques of first-order logic and information about $HL(\Sigma, T)$.

The theorem can be stated differently as follows:

$$PC(AX[\mathrm{Mod}(\Sigma, T)]) = HL(\Sigma, T).$$

Proof. First, we consider soundness: suppose that $HL(\Sigma, T) \vdash \{p\} S \{q\}$. Let $A \in \mathrm{Mod}(\Sigma, T)$ and let $(\sigma, \tau) \in AX(\Sigma, T)_A(S)$; we must show that $A \models p(\sigma)$ implies $A \models q(\tau)$. Now this follows immediately from the definition of $AX(\Sigma, T)_A(S)$ and our assumption that $HL(\Sigma, T) \vdash \{p\}\, S\, \{q\}$.

For completeness we assume, contrapositively, that $HL(\Sigma, T) \nvdash \{p\}\, S\, \{q\}$ and show that $AX(\Sigma, T) \not\models \{p\}\, S\, \{q\}$; the latter means that there exists $A \in \mathrm{Mod}(\Sigma, T)$ and $(\sigma, \tau) \in AX(\Sigma, T)_A(S)$ for which $A \models p(\sigma)$ does *not* imply $A \models q(\tau)$. The con-

struction of this interpretation A and input-output pair (σ, τ) is a lengthy exercise in logic and is organised by Lemmas 4.2 and 4.3 below.

Let $x = x_1, \ldots, x_k$ be a list of all free variables in p, q and all variables of S. We choose a list $c = c_1, \ldots, c_k$ of new constant symbols and add it to Σ to make $\Sigma_c = \Sigma \cup \{c_1, \ldots, c_k\}$. Let $p(c)$ be the sentence of $L(\Sigma_c)$ obtained by substituting each c_i for x_i in p $(1 \leqq i \leqq k)$; and set $T_p = T \cup \{p(c)\}$. A structure $A_0 \in \mathrm{Mod}(\Sigma_c, T_p)$ consists of a structure $A \in \mathrm{Mod}(\Sigma, T)$ augmented by an input $a = a_1, \ldots, a_k \in A$ that satisfies p. We write $A_0 = (A, a)$ and $A = A_0|_\Sigma$ in conventional reduct notation. Notice that $\mathrm{State}(A) = \mathrm{State}(A_0)$.

Let $x = c$ abbreviate $\bigwedge_{i=1}^{k} x_i = c_i$.

4.2. Lemma. $HL(\Sigma_c, T_p) \nvdash \{x = c\}\, S\, \{q\}$.

4.3. Lemma. *There is* $A_0 \in \mathrm{Mod}(\Sigma_c, T_p)$ *such that* $AX(A_0) \not\models \{x = c\}\, S\, \{q\}$.

Using these lemmas we can finish the proof of Theorem 4.1 as follows:

Let A be the Σ-reduct $A_0|_\Sigma$ so that $A \in \mathrm{Mod}(\Sigma, T)$. By Corollary 3.8, to show that $AX(\Sigma, T) \not\models \{p\}\, S\, \{q\}$ it is sufficient to show that $AX(A) \not\models \{p\}\, S\, \{q\}$.

By the description of A_0 in Lemma 4.3, there exists $(\sigma, \tau) \in AX(A_0)(S)$ such that $A_0 \models [x = c](\sigma)$ but $A_0 \not\models q(\tau)$. Since $A_0 \models p(c)$ we deduce that $A_0 \models p(\sigma)$ and because $p \in L(\Sigma)$ this implies that $A \models p(\sigma)$. In addition, we note that $A_0 \not\models p(\tau)$ implies that $A \not\models q(\tau)$ because $q \in L(\Sigma)$. Hence, we have demonstrated that:

$$A \models p(\sigma) \quad \text{does not imply} \quad A \models q(\tau).$$

It remains to remark that $(\sigma, \tau) \in AX(A)(S)$ by Lemma 3.9 and hence that $\{p\}\, S\, \{q\}$ is not partially correct with respect to $AX(A)$.

Proof of Lemma 4.2. Suppose for a contradiction that $HL(\Sigma_c, T_p) \vdash \{x = c\}\, S\, \{q\}$. By the Deduction Lemma 2.7,

$$HL(\Sigma_c, T) \vdash \{x = c \wedge p(c)\}\, S\, \{q\}$$

By Lemma 2.8, we may replace the constants $c = c_1, \ldots, c_k$ by new variables $y = y_1, \ldots, y_k$ not in p, q or S and obtain

$$HL(\Sigma, T) \vdash \{x = y \wedge p(y)\}\, S\, \{q\}.$$

By the $\exists$-Rule 2.5,

$$HL(\Sigma, T) \vdash \{\exists y (x = y \wedge p(y))\}\, S\, \{\exists y\, .\, q\}.$$

Applying the Rule of Consequence on this specified program using

$$T \vdash p(x) \to \exists y (x = y \wedge p(y)) \quad \text{and} \quad T \vdash \exists y\, .\, q \to q$$

we deduce that

$$HL(\Sigma, T) \vdash \{p\}\, S\, \{q\}$$

which is a contradiction. □

Proof of Lemma 4.3. Let $\{(r_i, t_i) : i \in \omega\}$ be an enumeration of all assertions of $L(\Sigma)$ such that $HL(\Sigma_c, T_p) \vdash \{r_i\}\, S\, \{t_i\}$. Let us write $r_i = r_i(x, z_i)$ and $t_i = t_i(x, z_i)$ where z_i lists all free variables in r_i and t_i distinct from those in the list x.

Define $\phi_i(x) = \forall z_i[r_i(c, z_i) \to t_i(x, z_i)]$ wherein $r(c, z_i)$ is $r(x, z_i)$ with c_i replacing x_i $(1 \leqq i \leqq k)$.

4.4. Lemma. *For no $n \in \omega$ is it the case that*

$$T_p \vdash \bigwedge_{i=1}^{n} \phi_i(x) \to q$$

Proof. Suppose for a contradiction that

$$T_p \vdash \bigwedge_{i=1}^{m} \phi_i(x) \to q \tag{1}$$

for some $m \in \omega$. We will prove that for each $i \in \omega$

$$HL(\Sigma_c, T_p) \vdash \{x = c\}\, S\, \{\phi_i(x)\}. \tag{2}$$

Then by the Conjunction Rule Lemma 2.2,

$$HL(\Sigma_c, T_p) \vdash \{x = c\}\, S \left\{ \bigwedge_{i=1}^{m} \phi_i(x) \right\}$$

and by the Rule of Consequence, using (1), we obtain

$$HL(\Sigma_c, T_p) \vdash \{x = c\}\, S\, \{q\}$$

which contradicts Lemma 4.2. Thus, it is sufficient to deduce (2) as a general fact.

From the enumeration we may assert that

$$HL(\Sigma_c, T_p) \vdash \{r_i(x, z_i)\}\, S\, \{t_i(x, z_i)\}.$$

By the Invariant Rule 2.3, since the variables of z_i are distinct from those of S,

$$HL(\Sigma_c, T_p) \vdash \{\neg r_i(c, z_i)\}\, S\, \{\neg r_i(c, z_i)\}.$$

By the Disjunction Rule Lemma 2.1,

$$HL(\Sigma_c, T_p) \vdash \{r_i(x, z_i) \vee \neg r_i(c, z_i)\}\, S\, \{t_i(c, z_i) \vee \neg r_i(x, z_i)\}.$$

By the Rule of Consequence

$$HL(\Sigma_c, T_p) \vdash \{x = c \wedge (r_i(x, z_i) \vee \neg r_i(c, z_i))\}\, S\, \{r_i(x, z_i) \to t_i(c, z_i)\}$$

and by a further application we can simplify the specified program to

$$HL(\Sigma_c, T_p) \vdash \{x = c\}\, S\, \{r_i(x, z_i) \to t_i(c, z_i)\}$$

By the $\forall$-Rule Lemma 2.4,

$$HL(\Sigma_c, T_p) \vdash \{x = c\}\, S\, \{\forall z_i[r_i(c, z_i) \to t_i(x, z_i)]\}$$

and by the definition of $\phi_i(x)$ this statement (2). □

4.5. Lemma. *There is* $B \in \mathrm{Mod}(\Sigma_c, T_p)$ *such that for some* $b = b_1, \ldots, b_k \in B$ *and for each* $i \in \omega$

$$B \models \phi_i(b) \quad \textit{but} \quad B \not\models q(b).$$

Proof. Suppose the lemma is false. Let $d = d_1, \ldots, d_k$ be a list of new constants and adjoin d to Σ_c to make $\Sigma_{c,d}$. Define the set

$$\Gamma = \{\phi_i(d) : i \in \omega\} \cup \{\neg q(d)\}$$

wherein $\phi_i(d)$ and $q(d)$ are ϕ_i and q with x_i replaced by d_i $(1 \leqq i \leqq k)$. Our supposition asserts that the set of sentences $T_p \cup \Gamma$ has no model and is therefore inconsistent. By the Compactness Theorem, some finite subset Γ_0 of $T_p \cup \Gamma$ is inconsistent. This means that for some $n \in \omega$ dependent on Γ_0,

$$T_p \vdash \bigwedge_{i=1}^{m} \phi_i(d) \to q(d).$$

On replacing the constants d by the variables of x we obtain a contradiction with Lemma 4.4. □

We claim that the Structure B in Lemma 4.5 can serve as the structure A_0 required in Lemma 4.3

Let $\sigma, \tau \in \mathrm{State}(B)$ such that

(i) $\sigma(v) = \tau(v)$ for each $v \notin x$;

(ii) $\sigma(x_i) = a_i$ where a_i is named by c_i

(iii) $\tau(x_i) = b_i$ where b_i is given in Lemma 4.5.

By Lemma 4.5, $B \models p(\sigma)$ does not imply $B \models q(\tau)$; and so we have to show that $(\sigma, \tau) \in AX(B)(S)$.

By Theorem 3.7, $AX(B)(S) = AX(\Sigma_c, T_p)_B(S)$. Suppose $r, t \in L(\Sigma)$ are such that $HL(\Sigma_c, T_p) \vdash \{r\} S \{t\}$. Then for some $i \in \omega$, $r = r_i$ and $t = t_i$ in our enumeration. Because $B \models \phi_i(b)$ we know that

$$B \models \forall z_i (r(a, z_i) \to t(b, z_i)).$$

By (i) we can take $\sigma(z_i) = \tau(z_i) = e$ for some list of elements e and write

$$B \models r(c, e) \to t(b, e).$$

Hence $B \models r(\sigma)$ implies $B \models t(\tau)$ and we may conclude that $(\sigma, \tau) \in AX(\Sigma_c, T_p)_B(S)$.

This ends the proof of Lemma 4.3 and the argument for completeness. □

Finally, notice that Theorem 4.1 has this to say about computation on an individual structure:

4.6. Corollary. *Let* A *be a structure of signature* Σ. *For each program* $S \in WP(\Sigma)$ *and any assertions* $p, q \in L(\Sigma)$, $HL(A) \vdash \{p\} S \{q\}$ *if, and only if, for every structure* B *elementarily equivalent to* A *we have* $AX(B) \models \{p\} S \{q\}$.

The corollary is proved by simply taking $T = Th(A)$ in Theorem 4.1. Notice, in particular, that it does not answer the following

4.7. Question. *Given any structure A, is it the case that*

$$HL(A)\vdash\{p\}\,S\,\{q\} \quad \textit{if, and only if,} \quad AX(A)\models\{p\}\,S\,\{q\}$$

for any $p, q\in L(\Sigma)$ and $S\in WP(\Sigma)$?

This question will be answered in Section 7.

5. Comparison of Axiomatic and Operational Semantics

To contrast the axiomatic and operational semantics we prove some theorems about their effects on computation on an individual structure A.

5.1. Theorem. *For any structure A of signature Σ and any program $S\in WP(\Sigma)$,*

$$OP(A)(S)\subseteq AX(A)(S).$$

Proof. Suppose $(\sigma,\tau)\in OP(A)(S)$. Now to show that $(\sigma,\tau)\in AX(A)(S)$ is to show for any assertions $p, q\in L(\Sigma)$ if $HL(A)\vdash\{p\}\,S\,\{q\}$ then $A\models p(\sigma)$ implies $A\models q(\tau)$. But this is an immediate consequence of the soundness of Hoare's logic $HL(A)$ for the operational semantics $OP(A)$. □

This theorem is a special case of the following more general fact:

$$HL(A)\subseteq PC(M_A) \quad \text{implies} \quad M_A(S)\subseteq AX(A)(S)$$

(the special case is obtained by taking $M_A=OP_A$, and observing $HL(A)\subseteq SPC(A)=PC(OP_A)$).

Theorem 5.1 means that if τ can be computed as *the* output state from input state σ using S under its standard operational semantics then τ can be obtained as *an* output state from σ using S and its axiomatic semantics: whilst $OP(A)(S)$ is deterministic, we do not know that $AX(A)(S)$ is deterministic.

5.2. Lemma. *Let A be a structure of signature Σ and $S\in WP(\Sigma)$. Suppose $\phi: A\to A$ is an automorphism of A. Then $(\sigma,\tau)\in AX(A)(S)$ implies $(\sigma,\phi(\tau))\in AX(A)(S)$.*

Proof. States τ and $\phi(\tau)$ cannot be distinguished by any formula; hence $(\sigma,\phi(\tau))$ will be in $AX(A)(S)$ just as (σ,τ) is.

5.3. Theorem. *There is a finite structure A for which*

$$OP(A)\neq AX(A)$$

Proof. Consider the two element structure $A=\{0,1\}$ having no functions, relations or constants.

Let $\phi(0)=1$, $\phi(1)=0$. Thus ϕ is an automorphism of A. It follows by the previous lemma that $AX(A)$ is not deterministic, and thereby differs from $OP(A)$.

5.4. Remark. It is remarkable that the structure A of 5.3 is expressive (as it is finite) but still shows a pathology concerning program semantics.

Moreover, one may notice that in A the axiomatic semantics does not capture the rather basic intuition of *sequential* program execution which asserts that program execution has no *side effects* on variables that do not occur in the program. (First order Hoare logic seems not to capture this intuition.)

Notice that the structure A is not minimal in the sense of Section 1.6. We will next consider minimal structures; we first show that in this case side effects on non program variables are absent.

5.5. Theorem. *Let A be a minimal structure of signature Σ. Suppose $x \in \mathrm{Var}(S)$ and $(\sigma, \tau) \in AX(A)(S)$. Then $\sigma(X) = \tau(X)$.*

Proof. Let $A \models [x=t](\sigma)$ with t a closed term. One easily proves with induction on the structure of S that $HL(A) \vdash \{x=t\}\, S\, \{x=t\}$. Therefore $A \models [x=t](\tau)$; hence $\tau(x) = \sigma(x)$.

5.6. Theorem. *Let A be a minimal structure of signature Σ and $S \in WP(\Sigma)$. If $(\sigma, \tau) \in OP(A)(S)$ and $(\sigma, \tau') \in AX(A)(S)$ then $\tau = \tau'$.*

Proof. Let x be a variable, we prove that $\tau(x) = \tau'(x)$. Let $x_1, \ldots, x_k$ be the variables occurring in S and choose closed terms $t_1, \ldots, t_k$, t such that $A \models [x_i = t_i](\sigma)$ and $A \models [x=t](\sigma)$. Now choose closed terms v_i, v such that $A \models [x_i = v_i](\tau)$ and $A \models [x=v](\tau)$. Observe that

$$OP(A) \models \left\{ \bigwedge_{i=1}^{k} x_i = t_i \wedge x = t \right\} S \left\{ \bigwedge_{i=1}^{k} x_i = v_i \wedge x = v \right\}.$$

By Lemma 2.12, we may deduce that

$$HL(A) \vdash \left\{ \bigwedge_{i=1}^{k} x_i = t_i \wedge x = t \right\} S \left\{ \bigwedge_{i=1}^{k} x_i = v_i \wedge x = v \right\}$$

It follows that $A \models \left[\bigwedge_{i=1}^{k} x_i = v_i \wedge x = v \right](\tau')$ and in particular, $A \models [x=v](\tau')$ whence $\tau'(x) = \tau(x)$ in A.

Thus, on the *operational domain* of S on A

$$DOM_A(S) = \{\sigma \in \mathrm{State}(A) : \exists \tau \in \mathrm{State}(A).\, (\sigma, \tau) \in OP(A)(S)\}$$

the axiomatic semantics is single-valued and coincides with the operational semantics; we record this fact as follows:

5.7. Corollary. *For any minimal structure A of signature Σ and any program $S \in WP(\Sigma)$, the operational semantics $OP(A)(S)$ is faithfully embedded in the axiomatic semantics $AX(A)(S)$.*

5.8. Corollary. *For any minimal structure A of signature Σ and any program $S \in WP(\Sigma)$ if S is everywhere convergent under its operational semantics then*

$$OP(A)(S) = AX(A)(S)$$

In particular, if there is a difference between the axiomatic and operational semantics of a program S it must arise at an input state where S fails to

converge to an output state under its conventional operational semantics; furthermore, at such an input, S may converge to many output states under its axiomatic semantics.

The next result recovers Cook's analysis of completeness: recall Theorem 2.10:

5.9. Theorem. *Let A be a minimal structure of signature Σ and suppose that the assertion language $L(\Sigma)$ is expressive for $WP(\Sigma)$ with respect to $OP(A)$. Then for every $S \in WP(\Sigma)$*

$$OP(A)(S) = AX(A)(S)$$

and the axiomatic semantics is deterministic. Moreover, $HL(A)$ is sound and complete for $AX(A)$: for any $p, q \in L(\Sigma)$,

$$HL(A) \vdash \{p\}\, S\, \{q\} \quad \textit{if, and only if,} \quad AX(A) \models \{p\}\, S\, \{q\}.$$

Proof. Everything in the theorem follows from equality of the two semantics. By Theorem 5.1, it is sufficient to show that $AX(A)(S) \subseteq OP(A)(S)$.

Suppose $(\sigma, \tau) \in AX(A)(S)$. If $O_A(\sigma)\downarrow\tau'$ then by Theorem 5.3, $\tau = \tau'$ and $(\sigma, \tau) \in OP(A)(S)$. Now assume $O_A(\sigma)\uparrow$; let $x_1, \ldots, x_k$ be the variables of S and let $t_1, \ldots, t_k$ be closed terms such that $\sigma(x_i) = t_i$. We find that

$$OP(A) \models \left\{ \bigwedge_{i=1}^{k} x_i = t_i \right\} S \{\text{false}\}$$

and, by Theorem 2.10,

$$HL(A) \vdash \left\{ \bigwedge_{i=1}^{k} x_i = t_i \right\} S \{\text{false}\}.$$

It follows that $A \models [\text{false}]\ (\tau)$ (by definition of $AX(A)(S)$. This is a contradiction which concludes the proof. □

5.10. Corollary. *Let $\mathbb{N}$ be the standard model of arithmetic with signature Σ, namely*

$$\mathbb{N} = (\{0, 1, \ldots\}: 0, x+1, x+y, x \,.\, y).$$

Then for every **while***-program $S \in WP(\Sigma)$*

$$OP(\mathbb{N})(S) = AX(\mathbb{N})(S).$$

Theorem 5.9 allows us to prove the following curious characterisation of the operational semantics.

5.11. Theorem. *For any structure of signature Σ and any program $S \in WP(\Sigma)$,*

$$OP(A)(S) = \cap\{AX(B)(S): B \textit{ is an expansion of } A\}$$

Proof. If B is an expansion of A then for $S \in WP(\Sigma)$

$$OP(A)(S) = OP(B)(S) \subseteq AX(B)(S)$$

by Theorem 5.1; this proves one inclusion. For the reverse inclusion, choose a structure B which is minimal and is an expansion of A and is expressive with respect to $OP(B)$: such B can be made by adding appropriate arithmetic coding functions following the methods in Section 3 of [12]. Then by Theorem 5.6 we have

$$AX(B)(S) = OP(B)(S) = OP(A)(S). \quad \square$$

6. Axiomatic Semantics and Computability

By Corollary 5.10, the functions and relations computable by the operational and axiomatic semantics coincide on the standard model of arithmetic $\mathbb{N}$; in particular, the class of partial functions on $\mathbb{N}$ definable by **while**-programs under their axiomatic semantics is the class of partial recursive functions. More generally, we may observe, using the definitions of 1.6, and Theorem 5.9, that

6.1. Lemma. *Let A be a minimal computable structure of signature Σ and suppose that $L(\Sigma)$ is expressive for $WP(\Sigma)$ with respect to $OP(A)$. Then every partial function on A definable by a* **while***-program under its axiomatic semantics is computable.*

The converse is not true because of the remark following Lemma 1.7.

6.2. Question. *What sets of functions are definable by* **while***-programs under axiomatic semantics on computable, but non-expressive, structures such as Presburger arithmetic*

$$P = (\{0, 1, \ldots\}: 0, x+1)?$$

The set of **while**-programs under operational semantics defines the set of partial recursive functions on Presburger arithmetic. The main task of this section is to show that the axiomatic semantics is able to define non-recursive functions (Corollary 6.5).

6.3. Theorem. *Let A be a minimal effectively enumerated structure of signature Σ and let $S \in WP(\Sigma)$. Then*

(1) $OP(A)(S)$ is recursively enumerable in $Th(A)$; and

(2) $AX(A)(S)$ is co-recursively enumerable in $Th(A)$.

Therefore, if $AX(A)(S) = OP(A)(S)$ then the set is decidable in $Th(A)$.

Proof. The proof of statement (1) is left as an exercise. Consider the definition of $AX(A)(S) \subset \text{States}(A)$:

$$\begin{aligned}(\sigma, \tau) \in AX(A)(S) &\Leftrightarrow (\forall p, q \in L(\Sigma))[HL(A) \vdash \{p\}\, S\, \{q\} \Rightarrow A \models p(\sigma) \to q(\tau)] \\ &\Leftrightarrow (\forall p, q \in L(\Sigma))[\{p\}\, S\, \{q\} \in HL(A) \Rightarrow p(\sigma) \to q(\tau) \in Th(A)] \\ &\Leftrightarrow (\forall p, q \in L(\Sigma))[\{p\}\, S\, \{q\} \notin HL(A) \vee p(\sigma) \to q(\tau) \in Th(A)]\end{aligned}$$

With an enumeration of A, a derived codification of States(A), a gödel numbering of $L(\Sigma)$, and derived codifications of $Th(A)$ and $HL(A)$, we may formally express the fact that $\{p\}\, S\, \{q\} \notin HL(A)$ is co-recursively enumerable in

$Th(A)$, and hence deduce that $AX(A)(S)$ is co-recursively enumerable in $Th(A)$. □

6.4. Corollary. *Let A be a minimal computable structure and suppose $Th(A)$ is decidable. If $AX(A)(S)=OP(A)(S)$ then the set is decidable. In particular, if A is expressive then for every S the set $AX(A)(S)=OP(A)(S)$ is decidable.*

6.5. Corollary. *There is a* **while***-program S over Presburger arithmetic P such that*

$$OP(P)(S) \subsetneqq AX(P)(S)$$

Proof. P satisfies the hypotheses of Corollary 6.3. Each recursively enumerable set Ω may be represented as the graph of a partial recursive function: let S be a program such that

$$\Omega = OP(P)(S)$$

where Ω is an r.e. non-recursive set. □

7. Completeness for Structures

Question 4.7 asked if $HL(A)$ is complete for the axiomatics $AX(A)$ for any structure A. Here is the answer:

7.1. Theorem. *There is a structure A and specified program $\{p\}\,S\{q\}$ such that*

$$AX(A) \models \{p\}\,S\{q\} \quad \textit{but} \quad HL(A) \not\vdash \{p\}\,S\{q\};$$

in particular, $HL(A)$ is not complete for $AX(A)$.

Sketch Proof. Let $A=(\{0,1,\ldots\}: 0, x+1, x\dot{-}1)$ and consider the program

$$S :: = x := 0; \quad \textbf{while } y \neq 0 \textbf{ do } x := x+1; \quad y := y \dot{-} 1 \textbf{ od}.$$

This S with p and q defined by

$$y = z \quad \text{and} \quad x = z$$

respectively forms the specified program required in the theorem. The validity property, that $AX(A) \models \{p\}\,S\{q\}$, follows from Corollary 5.7, because S is everywhere convergent under $OP(A)$. The fact that $\{p\}\,S\{q\}$ cannot be derived in $HL(A)$ is more complicated to prove. It begins with the observation that an invariant for the loop in S would define addition $x+y$ on A. However one can prove that A admits quantifier elimination (since A is so close to Presburger Arithmetic P); and further that boolean or quantifier-free formulae over A cannot define the graph of addition on A. Thus no invariant for the loop can exist, and hence no deduction of $\{p\}\,S\{q\}$ in $HL(A)$. □

The result should be compared with Theorem 5.9; we conclude this section with two further results on completeness.

7.2. Lemma. *For any structure A, if $HL(A)$ is complete for the operational semantics $OP(A)$ then it is complete for the axiomatic semantics $AX(A)$.*

Proof. Suppose that $HL(A)$ is complete for $OP(A)$ and assume that $AX(A) \models \{p\}\, S\, \{q\}$. By Theorem 5.1, $OP(A) \models \{p\}\, S\, \{q\}$ and, therefore, $HL(A) \vdash \{p\}\, S\, \{q\}$. □

7.3. Theorem. *There is a structure A such that $HL(A)$ is complete for $AX(A)$ but $HL(A)$ is not complete for $OP(A)$.*

Proof. Again we outline the argument which is based on our earlier studies of Hoare's logic and arithmetic. The following result is rather interesting:

7.4. Theorem. *Let A be a model of Peano Arithmetic PA. Then $HL(A)$ is complete for $AX(A)$. Furthermore, for each $S \in WP(\Sigma_{PA})$, $AX(A)(S)$ is first-order definable over A.*

Sketch Proof. The proof is based upon the principal theorem in [12] which establishes a strongest postcondition calculus for Peano Arithmetic: for each precondition p and program S there is a first-order formula $SP(p, S)$ such that for each postcondition q

$$HL(PA) \vdash \{p\}\, S\, \{q\} \quad \text{if, and only if,} \quad PA \vdash SP(p, S) \rightarrow q$$

and, in particular,

$$HL(PA) \vdash \{p\}\, S\, \{SP(p, S)\}. \quad \square$$

We couple with Theorem 7.4 one of the main theorems in [10], namely: for a model A of Peano Arithmetic, $HL(A)$ is complete for $OP(A)$ if, and only if, A is elementary equivalent to the standard model $\mathbb{N}$. Thus, to prove Theorem 7.3 it is sufficient to choose A to be any model of Peano Arithmetic that is not elementary equivalent to $\mathbb{N}$. □

8. Non-Determinism on Minimal Structures

The basic relationships existing between the axiomatic semantics and operational semantics were worked out in Sections 5 and 6. In this last section we examine in greater detail a principal semantic difference between the meanings: the axiomatic semantics need not be deterministic. We will construct a minimal structure A and a program S such that $AX(A)(S)$ is not a single-valued relation on A.

The program S has form

$$S ::= \textbf{while } b \textbf{ do } S_0 \textbf{ od}$$

where S_0 contains no loops: let us refer to such a program as being in *simple form*. The following is routine:

8.1. Lemma. *Let A be a structure of signature Σ and let $S \in WP(\Sigma)$ be in simple form. Then a sufficient condition for $(\sigma, \tau) \in AX(A)(S)$ is the following: for each invariant $I \in L(\Sigma)$ where*

$$HL(A) \vdash \{I \wedge b\}\, S_0 \{I\}$$

we have that

$$A \models I(\sigma) \quad \textit{implies} \quad A \models (I \wedge \neg b)(\tau)$$

In applying the lemma to two distinct pairs (σ, τ) and (σ, τ'), and thereby demonstrating nondeterminism, the sufficient condition can be verified by means of the following condition:

8.2. Patching Lemma. *A sufficient condition for* $(\sigma, \tau) \in AX(A)(S)$ *is that* $A \models b(\sigma)$, $A \models \neg b(\tau)$ *and for each invariant* I *such that* $A \models I(\sigma)$ *there is a pair of sequences of states*

$$\sigma = \sigma_0, \sigma_1, \ldots, \sigma_k \quad \textit{and} \quad \tau_0, \tau_1, \ldots, \tau_l = \tau$$

for which

(i) $A \models b(\sigma_j)$ *for* $0 \leqq j \leqq k$ *and* $A \models b(\tau_j)$ *for* $0 \leqq j < l$

(ii) $(\sigma_i, \sigma_{i+1}) \in OP(A)(S_0)$ *and* $(\tau_j, \tau_{j+1}) \in OP(A)(S_0)$ *for* $0 \leqq i \leqq k-1$ *and* $0 \leqq j \leqq l-1$;

(iii) $A \models I(\sigma_k)$ *implies* $A \models I(\tau_0)$.

To see this suppose $HL(A) \vdash \{I \wedge b\} S_0 \{I\}$, $A \models I(\sigma)$, $A \models b(\sigma)$, and $A \models \neg b(\tau)$.

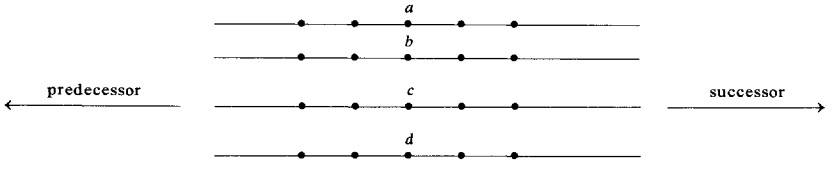
It follows that $A \models (I \wedge b)(\sigma_j)$ for $1 \leqq j \leqq k$ by induction on j. $I(\sigma_k)$ implies $I(\tau_0)$ thus $A \models (I \wedge b)(\tau_0)$; with induction on $j < l$ we find $A \models (I \wedge b)(\tau_j)$ from which $A \models (I \wedge \neg b)(\tau)$.

The condition describes how two operational computations can be "patched" by the formula I. We can now proceed with the example.

8.3. Example. Let $\Sigma = \{a, b, c, d, SUCC, PRED\}$ where a, b, c, d, are constant symbols and $SUCC$ and $PRED$ are unary operator symbols. Let E be the set containing the equations

$$SUCC(PRED(X)) = X$$
$$PRED(SUCC(X) = X$$

We take A to be the initial algebra semantics of the algebraic specification (Σ, E): see *ADJ* [20]. The following picture may be helpful in understanding the structure A:



Next we define a program S by

$$S ::= \textbf{while } x \neq a \textbf{ do } x := PRED(x);\ y := SUCC(y) \textbf{ od}$$

And we define the states σ, τ, τ' by

$$\sigma(v) = \begin{cases} b & \text{if } v = x \\ a & \text{otherwise} \end{cases}$$

$$\tau(v)=\begin{cases} c & \text{if } v=y \\ a & \text{otherwise} \end{cases}$$

$$\tau'(v)=\begin{cases} d & \text{if } v=y \\ a & \text{otherwise} \end{cases}$$

8.4. Claim. *It is the case that* $(\sigma,\tau)\in AX(A)(S)$ *and* $(\sigma,\tau')\in AX(A)(S)$.

Proof. We show that $(\sigma,\tau)\in AX(A)(S)$; the other axiomatic computation is treated similarly.

With reference to the Patching Lemma 8.2, clearly

$$A\models\neg(x\neq a)(\tau)$$

Suppose that I is an invariant with

$$HL(A)\vdash\{I\wedge x\neq a\}\ x:=PRED(x);\ y:=SUCC(y)\{I\}$$

and $A\models I(\sigma)$; we must now construct appropriate sequences of states which serve as operational computations, of lengths k and l, patched by I.

In our construction $k=l$ and, postponing the calculation of the value of k, we define the states for $i=1,\ldots,k$

$$\sigma_i(v)=\begin{cases} PRED^i(b) & \text{if } v=x \\ SUCC^i(a) & \text{if } v=y \\ a & \text{otherwise} \end{cases}$$

$$\tau_0^{(k)}=\begin{cases} SUCC^k(a) & \text{if } v=x \\ PRED^k(c) & \text{if } v=y \\ a & \text{otherwise} \end{cases}$$

$$\tau_i^{(k)}=\begin{cases} PRED^iSUCC^k(a) & \text{if } v=x \\ SUCC^iPRED^k(c) & \text{if } v=y \\ a & \text{otherwise} \end{cases}$$

We take $\sigma_0=\sigma$ and note that $\tau_k^{(k)}=\tau$.

What remains is for us to show there is a value of k that is sufficiently large to ensure that

$$A\models I(\sigma_k)\quad\text{implies}\quad A\models I(\tau_0^{(k)})$$

The existence of a k_0 such that, for all $k>k_0$, the implication is true is demonstrated by a combinatorial argument based on the fact that A admits quantifier elimination.

9. Discussion

The axiomatic relational semantics $AX(A)$ constitutes the *maximal* relational semantics consistent with $HL(A)$. It is implicit in the idea of partial correctness

that one considers a maximal (rather than a minimal) interpretation of program behaviour.

The operational semantics $OP(A)$ corresponds to intuition about program behaviour. The extent to which $AX(A)$ approximates $OP(A)$ is summarised in the following theorem:

9.1. Theorem. (i) *Suppose the Σ structure A is both minimal and expressive for $OP(A)$ then*

$$OP(A)=AX(A)$$

(ii) *Neither minimality nor expressiveness alone is a sufficient condition for the coextensiveness of $OP(A)$ and $AX(A)$.*

Proof. (i) This follows from Lemma 5.9. (ii) In 5.3 we have an expressive but non minimal structure and in 8.3 we have a minimal but non-expressive structure, both have different OP and AX. □

The condition of expressiveness removes a shortcoming of the assertion language. The minimality condition however seems to remove the inability of HL to prove the absence of side effects of program execution, on variables not occurring in the program.

Concerning side effects two problems remain unsettled. Let M be a **while** program semantics for the class K of Σ structures.

Definition. (i) M *avoids side effects on* non-program variables if for all $A\in K$, $S\in WP(\Sigma)$, $(\sigma,\tau)\in M(A)(S)$ and $x\notin \mathrm{Var}(S)$, $\sigma(x)=\tau(x)$.

(ii) M *avoids side effects from* non-program variables if for all $A\in K$, $S\in WP(\Sigma)$, $(\sigma,\tau)\in M(A)(S)$ the following holds : for each τ' that corresponds with σ on all variables occurring in S there is τ' that corresponds with τ on all variables of S such that $(\sigma',\tau')\in M(A)(S)$.

Problem 1. Let K be the class of minimal Σ structures. Can one prove that AX avoids side effects *from* non-program variables?

Problem 2. Is there a semantics M for $K=\mathrm{Mod}(\Sigma,T)$ which satisfies the completeness theorem $PC(M[\mathrm{Mod}(\Sigma,T)])=HL(\Sigma,T)$ and which avoids both types of side effect.

10. Concluding Remarks

We have shown that the axiomatic semantics AX of **while**-programs is a non-standard semantics far removed from the standard semantics of WP. In general, AX is non-deterministic and it need not be computable: on the structure Presburger Arithmetic P it cannot be implemented, in principle. But there are structures (for example, the standard model of arithmetic $\mathbb{N}$) on which the axiomatic semantics and the operational semantics coincide. In the light of these and the other results, what conclusions on axiomatic semantics, and the role of verification in language design, can be found?

The situation reminds us of the non-standard models of Peano Arithmetic PA. The system PA is a fundamental formal system that fails to capture the

equally fundamental semantics $\mathbb{N}$ for which it was specifically designed. The system is no less important as a logical tool for the study of number theory and the non-standard models are now considered as indispensible tools for the analysis of the system. We consider the case of Hoare's logic to be the same: Hoare's logic is a basic tool for the study of program verification, and the study of the non-standard semantics for **while**-programs, the true semantics for the proof system, will be significant in understanding the system. Ultimately, the system's importance will be determined by its role in the theoretical and practical exploration of the following idea:

10.1. Logicist's Thesis. *What may be known about our computer programs is represented, and delimited, by what may be formally proved about programs in logical systems.*

In the case of proving program correctness, Hoare's logic, as we have defined it, maintains its eminence for three connected reasons. First, it is based on first-order logic which is the logical system known in greatest depth. Secondly, logical systems for total correctness are fraught with difficulties associated with the proof of termination; in addition, first-order logic cannot be used to specifiy termination (see Apt [1]). Thus, partial correctness is the principal property for which we can make and study formal systems. Thirdly, the first-order Hoare's logic for **while**-programs serves as the prototype for the manufacture of partial correctness logics for most of our contemporary programming languages and in these logics specific programs can be verified: see Apt [1] and McGettrick [31].

If one is interested in the Logicist Thesis then the Floyd-Hoare Principle is fundamentally important.

Finally, we will make references to research relevant to the central concerns of this paper.

A rather different perspective on completeness theorems, and hence on axiomatic semantics, emerges from the work of the Hungarian School of I. Nemeti, H. Andreka, I. Sain and L. Csirmaz on the logical foundations of verification [2, 3, 4, 15, 16]. Among the subjects they have investigated in great depth is completeness for first-order systems involving axiomatic time. Technically, their time structure semantics provides an alternative route to the completeness theorem we prove, but further work is required to reconcile their results with the questions examined here.

We note that Magidor and J. Stavi have made a completeness theorem for an iterative language and its first-order partial correctness logic using a semantics involving time (personal communication).

For relevant work on the completeness problem outside first-order Hoare's logic see Kröger [26] in which ω-rules are considered and the monograph [37] in which a many-sorted finite sequencing mechanism is allowed.

Acknowledgement. We are grateful to Peter Lauer for his encouraging reception and detailed criticism of a first draft of this paper. We thank our referees for invaluable comments on the second draft of this paper. And we we thank Ms. Judith Thursby for typing this manuscript.

References

1. Apt, K.R.: Ten years of Hoare's logic. A Survey: Part 1. ACM Trans. Progr. Lang. Syst. **3**, 431–483 (1981)
2. Andreka, H., Nemeti, I.: Completeness of Floyd Logic. Bull. Sect. Logic Wroclaw **7**, 115–121 (1978)
3. Andreka, H., Nemeti, I., Sain, I.: A complete logic for reasoning about programs via nonstandard model theory. Theor. Comput. Sci. **17**, 193–212 (1982)
4. Andreka, H., Nemeti, I., Sain, I.: A characterization of Floyd-provable programs. In: Mathematical Foundations of Computer Science 1981, Lecture Notes in Computer Science 118. Berlin, Heidelberg, New York: Springer, pp. 162–171, 1981
5. deBakker, J.W.: Mathematical theory of program correctness. London: Prentice-Hall 1980
6. Bergstra, J.A., Tiuryn, J., Tucker, J.V.: Floyd's Principle, correctness theories and program equivalence. Theor. Comput. Sci. **17**, 113–149 (1982)
7. Bergstra, J.A., Tucker, J.V.: Some natural structures which fail to possess a sound and decidable Hoare-like logic for their while-programs. Theor. Comput. Sci. **17**, 303–315 (1982)
8. Bergstra, J.A., Tucker, J.V.: Algebraically specified programming systems and Hoare's logic. In: International Colloquium on Automata, Languages and Programming 1981, Lecture Notes in Computer Science 115. Berlin, Heidelberg, New York: Springer pp. 348–362, 1981
9. Bergstra, J.A., Tucker, J.V.: The refinement of specifications and the stability of Hoare's logic. In: Logics of Programs, Lecture Notes in Computer Science 131. Berlin, Heidelberg, New York: Springer pp. 24–36, 1981
10. Bergstra, J.A., Tucker, J.V.: Expressiveness and the completeness of Hoare's logic. J. Comput. Syst. Sci. **25**, 276–284 (1982)
11. Bergstra, J.A., Tucker, J.V.: Two theorems about the completeness of Hoare's logic. Information Processing Lett. **15**, 143–149 (1982)
12. Bergstra, J.A., Tucker, J.V.: Hoare's logic and Peano's Arithmetic. Theor. Comput. Sci. **22**, 265–284 (1983)
13. Bergstra, J.A., Tucker, J.V.: Hoare's logic for programming languages with two data types. Theor. Comput. Sci. **28**, 213–221 (1984)
14. Cook, S.A.: Soundness and completeness of an axiomatic system for program verification. SIAM J. Comput. **7**, 70–90 (1978), Corrigendum **10**, 612 (1981)
15. Csirmaz, L.: On the completeness of proving partial correctness. Acta Cybernet. **5**, 181–190 (1981)
16. Csirmaz, L., Paris, J.B.: A property of 2-sorted Peano models and program verification. Preprint, Math. Inst., Hungarian Academy, Budapest, 1981
17. Dijkstra, E.W.: Guarded commands, nondeterminacy and formal derivation of programs. Commun. ACM **18**, 453–457 (1975)
18. Dijkstra, E.W.: A discipline of programming. Engelwood Cliffs: Prentice-Hall, 1976
19. Floyd, R.W.: Assigning meaning to programs. In: Mathematical aspects of computer science. J.T. Schwartz (ed.). AMS, pp. 19–32, 1967
20. Goguen, J.A., Thatcher, J.W., Wagner, E.G.: An initial algebra approach to the specification, correctness and implementation of abstract data types. In: Current trends in programming methodology IV, Data structuring. R.T. Yeh (ed.). Englewood Cliffs, New Jersey: Prentice-Hall, pp. 80–149, 1978
21. Greif, I., Meyer, A.R.: Specifying the semantics of while programs: a tutorial and critique of a paper by Hoare and Lauer. ACM Trans. Progr. Lang. Syst. **3**, 484–507 (1981)
22. Hoare, C.A.R.: An axiomatic basis for computer programming, Commun. ACM **12**, 576–580 (1969)
23. Hoare, C.A.R.: Procedures and parameters: an axiomatic approach. In: Symposium on the semantics of algorithmic languages. E. Engeler (ed.). Berlin, Heidelberg, New York: Springer pp. 102–116, 1971
24. Hoare, C.A.R., Lauer, P.: Consistent and complementary formal theories of the semantics of programming languages. Acta Informat. **3**, 135–155 (1974)
25. Hoare C.A.R., Wirth, N.: An axiomatic definition of the programming language PASCAL. Acta Informat. **2**, 335–355 (1973)

26. Kröger, F.: Infinite proof rules for loops. Acta Informat. **14**, 371–389 (1980)
27. Lauer, P.: Consistent and complementary formal theories of the semantics of programming languages, Ph. D. Thesis Queens University, Belfast, 1972
28. London, R.L., Guttag, J.V., Horning, J.J., Lampson, B.W., Mitchell, J.G., Popek, G.L.: Proof rules for the programming language EUCLID. Acta Informat. **10**, 1–26 (1978)
29. Manna, Z.: The correctness of programs. J. Comput. Syst. Sci. **3**, 119–127 (1969)
30. Mal'cev, A.I.: Constructive algebras I. Russian Math. Surveys **16**, 77–129 (1961)
31. McGettrick, A.: The definition of programming languages, Cambridge: University Press, 1980
32. Meyer, A.R., Halpern, J.Y.: Axiomatic definitions of programming languages. A theoretical assessment. J. ACM **29**, 555–576 (1982)
33. Meyer, A.R., Halpern, J.Y.: Axiomatic definitions of programming languages II. MIT Lab. Comput. Sci. TM-179 (1980)
34. Pagan, F.G.: Formal specification of programming languages. Englewood Cliffs: Prentice-Hall, 1981
35. Rabin, M.O.: Computable algebra, general theory and the theory of computable fields. Trans. AMS **95**, 341–360 (1960)
36. Schwartz, R.: An axiomatic semantic definition of ALGOL 68. UCLA Computer Science Report 7838, 1978
37. Tucker, J.V., Zucker, J.I.: Program correctness over abstract data types, with error state semantics, Monograph (In prep.)

Received July 8, 1983/July 9, 1984