stichting
mathematisch
centrum

MC

AFDELING INFORMATICA
(DEPARTMENT OF COMPUTER SCIENCE)

IW 145/80 SEPTEMBER

J.A. BERGSTRA, J. TIURYN & J.V. TUCKER

FLOYD'S PRINCIPLE, CORRECTNESS THEORIES AND PROGRAM EQUIVALENCE

Preprint

kruislaan 413 1098 SJ amsterdam

Printed at the Mathematical Centre, 413 Kruislaan, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

1980 Mathematics subject classification: 03C52, 03D75, 68B10

ACM - Computing Review-categories: 5.24

Floyd's principle, correctness theories and program equivalence *)

by

J.A. Bergstra **), J. Tiuryn ***) & J.V. Tucker

ABSTRACT

A programming system is a language made from a fixed class of data abstractions and a selection of familiar deterministic control and assignment constructs. It is shown that the sets of all "before-after" first-order assertions which are true of programs in any such language can uniquely determine the input-output semantics of the language providing one allows the use of auxiliary operators on its ground types.

After this, we study programming systems wherein the data types are syntactically defined using a first-order specification language with the objective of elimating these auxiliary operators. Especial attention is paid to algebraic specifications, complete first-order specifications; and to arithmetical computation in the context of a specified programming system.

KEY WORDS & PHRASES: *programming systems with and without data type specifications; program semantics specification; first-order correctness theories; program equivalence*

---

*) This report is an extensive revision of IW 119/79. It is not for review as it will be published elsewhere.

**) Department of Computer Science, University of Leiden, Wassenaarseweg 80, Postbus 9512, 2300 RA LEIDEN, The Netherlands

***) Mathematics Institute, University of Warsaw, PKiN 00-901 WARSAW, Poland

INTRODUCTION

The idea that a general programming language, or a specialised programming system, PS can be usefully defined by the axioms and rules of inference underlying proofs of various properties of programs written in the language can be traced to R.W. FLOYD [13]. As Hoare pointed out in [19], it is an attractive thesis for demanding that any implementation of PS be made to satisfy these axioms and rules provides a criterion for the correctness of its implementations and establishes a set of *provable* features for programs in PS common to all its implementations: everything proved true of a program S in PS will be true in each implementation however the all important undefined features of PS are handled. The acceptance of such a formal axiomatic system as authoritative in specifying the meaning of a language has been advocated by several writers. For example, E.W. DIJKSTRA [10], C.A.R. HOARE [20], Z. MANNA [23], HOARE and LAUER [21]; and, of course, in HOARE and WIRTH [22] where Floyd's idea is applied to make a definition for a part of PASCAL (see, in addition, R. SCHWARTZ [32]). Despite its familiarity in the literature on program language design, Floyd's thesis is, by the standards of the theoretical literature, as vague as it is intriguing: *what* proof systems for *which* properties of programs in *what* kinds of program languages can characterise semantics, and in *which* precise senses? It seems fair to say that, at present, as little is understood about the issues involved as was known about Hoare's logic for proving program correctness *before* S.A. Cook took up that subject in his seminal study [8] (see [9] and the important survey article of K.R. APT [1]).

This paper will settle upon one natural and precise formulation of Floyd's principle and will study it in quite some technical detail. The program properties on which the specification method is based we take to be *partial correctness* as this is formalised by first-order definable assertions. The semantics of a programming system we require to be defined uniquely up to the *input-output behaviour* of its programs, one of the standard measures of denotational semantics. Typically, we have in mind a programming system PS with all the usual deterministic control and assignment constructs and whose data types are fixed independently by, say, procedures without side effects in some general purpose programming language L. This

leads to a model of $\mathcal{PS}$ in which program texts are represented by program schemes of some standard design PROG - for example, while-programs with counters and stacks - and in which the data types are semantically given as various classes of interpretations for the primitives appearing in PROG. Each program S of $\mathcal{PS}$ will involve a finite collection Σ of constant and operator symbols and so will belong to the set PROG(Σ) of programs in PROG having this signature Σ. To PROG(Σ) is associated a class K of Σ-structures which we explicitly think of as representing the data type semantics of $\mathcal{PS}$ at least as far as the primitives in Σ are concerned.

We are thus led, in Section 2, to define various *partial correctness theories* $PC_K(S)$ as sets of first-order partial correctness assertions about S true throughout the interpretations of K. We shall say that a (particular kind of) *partial correctness theory determines the input-output semantics of the programming system* $\mathcal{PS}$ if for each pair of programs S, S' of $\mathcal{PS}$, admitting interpretations throughout K, if $PC_K(S) = PC_K(S')$ then S and S' compute the same partial function on each interpretation $A \in K$. And our paper will be taken up with investigating determinateness for various types of correctness theory and various classes of interpretations.

From the point of view of Floyd's thesis, we make rapid progress: our first theorem (Section 3, Theorem 3.1) does indeed confirm in a precise, formal and respectable way, that *there are first-order partial correctness theories available which determine the input-output semantics of any deterministic programming system.* As a result of this it can now be said that *as long as one can axiomatise the appropriate kind of correctness theory for a programming system* then Floyd's principle, in this formulation, is a theoretically realistic method for defining it. This proviso is, of course, the second half of the problem of making theoretical sense of Floyd's principle. And, as such, it is an almost independent source of many interesting problems about the existence, or non-existence, of sound and complete Hoare logics for proving the partial correctness of programs relative to given classes of interpretations. As will be made clear, the state of that particular art - as reported in Apt's recent survey [1], for example - is nowhere near sufficiently well developed to service this enquiry. Because of this, and other ramifications of the proviso, here we are content to concentrate on the determinateness problem and more or less ignore axiomatisations.

(Two of us have, however, begun to investigate Hoare logics for the partial correctness theories of highly specific programming systems [5] and we will provide here some information about how perplexing the situation seems to be.)

It now remains for us, in this introduction, to explain how a very general and reasonable solution to the determinateness problem fails to completely settle even that issue in Floyd's thesis and how it leads directly to the powerful machinery of algorithmic logic which characterises the second half of our paper. The correctness theories employed in our first solution have an irritating technical defect: they include assertions which use operators which do not appear in the programs and whose semantics are extrinsic to that of the programming systems as given. The problem of eliminating "hidden functions" from the specifying assertions forces us to be much more explicit about what the data type semantics of our programming systems really are and, in particular, how they are prescribed. In Section 5, we make the assumption that the classes of interpretations which represent the data type semantics of programming systems have specifications written out as axioms in the first-order assertion languages. And we explain how this hypothesis embraces the algebraic specification methods for data types (ADJ [15]) as well as the specification assumptions about data types which are implicit in studies of Hoare's logic in the manner of COOK [9]. After a change of correctness theory, we look at determinateness for programming systems with data type specifications and with a particular emphasis on those specifications used for algebraic definitions and used in the theory of Hoare logics. At this point, the model-theoretic nature of the problem of determinateness becomes clearly visible and its solution becomes the business of algorithmic logic. In Section 6, we tackle determinateness for computations on the set of natural numbers ω under the assumption that arithmetic is *not* semantically given outright, but must be syntactically specified as (part of) the semantics of a programming system.

Our interest in Floyd's thesis and its mathematical analysis we owe entirely to A.R. Meyer who invited us ([25]) to work on the problem of determinateness in cases of the form $K = ALG(\Sigma)$, the class of all structures of signature Σ, where implicitly no conditions are placed on the data types on which the programs compute. Subsequently, Meyer and J.Y. Halpern indepen-

dently, and exhaustively, analysed this special case ([26]) and a preliminary report on their work has already appeared [28]. All readers of this paper are recommended [28] for a detailed exposition of the informal issues involved in a theoretical examination of Floyd's thesis based upon the hypothesis that a general purpose program language should be modelled by program schemes PROG(Σ) and all interpretations ALG(Σ). This view is not, however, the outlook of the present paper. As will be explained in Section 2, here we take the (mathematically) more general notion of a programming system as the basic object of study and model a general purpose programming language by the totality of all possible programming systems. The advantage, as far as Floyd's principle is concerned, is a much sharper analysis of determinateness; the "disadvantage" is that the necessary layers of conceptual and technical complications ask as many new questions as they answer old ones. In any case, the mathematical results will speak for themselves: our readers should have no difficulties in comparing them with those of Meyer and Halpern and connecting them with the early work of DE BAKKER [2] (see also the more recent paper HENNESSY [18]) on proof systems for program equivalence; or, more generally, to the field of algorithmic logic associated with ENGELER [11], and the Polish and American Schools [4], [17].

1. PROGRAMS AND ASSERTIONS

Any of the common designs for deterministic program schemes will serve to model the programs required in our study: flow charts; while-programs; recursive procedures; all with, or without, arrays, counters, boolean variables and the like. This is because with input-output semantics what matters is the class of functions defined on an interpretation, not the mechanisms involved in their computation: *the meaning of a program is to be the mapping it computes*. If our work is to bear on Floyd's thesis then it is necessary (and, ultimately, it is sufficient) that we are able to consider program families which are sufficiently strong to compute on each interpretation A *all* those functions effectively calculable on A by means of finite deterministic algorithms. The appropriate generalised Church-Turing thesis is known: among many disparate, yet equivalent, definitions of computability on a relational structure A in use in the literatures of theoretical computer

science and mathematical logic, the formula which is most familiar to the reader is the set of all *flow charts with counters and arrays*. From the point of view of mathematical logic, the key characterisation is, perhaps, Y.N. Moschovakis' *absolute prime computability* in [30] as it *directly* defines the minimal model of the axiomatic notion of a computation theory over A, see J.E. Fenstad's monograph [12]. Between the two subjects lie the *effective definitional schemes* and *finite algorithmic procedures* of H. FRIEDMAN [14] which we, the authors, favour. The former concept is the one chosen for Meyer and Halpern's work [28], incidentally. (For a survey of research into the subject of a generalised Church-Turing thesis for general algebras see [29,34].)

The point is that with input-output semantics we can leave undefined the general class of deterministic programs PROG used throughout the paper thus allowing the reader to apply our results to the class(es) of his or her choice. In addition, the reader is free to choose the full computational semantics of his or her program formula from which its input-output semantics must be derived. The different ways of defining computational semantics are legion, of course: as well as the text-books on program schemes, MANNA [24] and GREIBACH [16], we recommend de Bakker's monograph on denotational semantics [3]. The interesting paper A. MEYER & I. GREIF [27] is useful for further guidance on issues involved in this "choice" of computational semantics.

To sum up, then, PROG represents some set of deterministic program schemes capable of defining all computable functions on any interpretation. We assume it closed under composition and if * then * else * fi statements and we will use it with the following notational conventions. The syntax of PROG has

$\underline{c}_1, \underline{c}_2, \ldots$	as constant symbols;
$^k f_1, {}^k f_2, \ldots$	as function symbols of k arguments;
$^k R_1, {}^k R_2, \ldots$	as relation symbols of k arguments;
$x_1, x_2, \ldots$	as variables.

But we reserve the right to abuse this notation by dropping the arities from function and relation symbols and by introducing y's and z's as

variables, and so on.

Each $S \in$ PROG is assumed to name certain variables as input variables and a variable as an output variable; this fixes the arity of the function S computes over its various interpretations. By the *signature* of $S \in$ PROG we mean the finite list $\Sigma(S)$ of all constant, function and relational symbols appearing in its text. The set of all $S \in$ PROG of signature Σ we denote by PROG(Σ). Thus, $S \in$ PROG(Σ) defines a partial function on precisely those relational structures whose signatures certain Σ. For such a program S and interpretation A, if S names n input variables and $a \in A^n$ then by S(a) we ambiguously denote the computation of S applied to $a \in A^n$ *and* the output value when this exists; converging and diverging computations are distinguished, as usual, by $S(a)\downarrow$ and $S(a)\uparrow$ respectively.

If P and Q are programs of signatures $\Sigma(P)$ and $\Sigma(Q)$ respectively, and A is a relational structure whose signature contains $\Sigma(P) \cup \Sigma(Q)$ then P and Q are said to be A-*equivalent*, written $P \equiv_A Q$, if for all $a \in A^n$ either $P(a)\downarrow$ and $Q(a)\downarrow$ and $P(a) = Q(a)$ or both $P(a)\uparrow$ and $Q(a)\uparrow$.

We take for granted that the reader has available, in his or her computational semantics for PROG, formal definitions of a *state description* in a computation S(a) and of *length of computation* $|S(a)|$ and that the following basic facts can be proved (see [34]):

1.1 LOCALITY OF COMPUTATION LEMMA. *In any computation* $S(a_1,\ldots,a_n)$ *all the elements of* A *appearing in all the state descriptions of* $S(a_1,\ldots,a_n)$ *lie within* $\langle a_1,\ldots,a_n\rangle$, *the subalgebra of* A *generated by* $a_1,\ldots,a_n \in A$. *In particular, if* $S(a_1,\ldots,a_n)$ *converges then its output lies in* $\langle a_1,\ldots,a_n\rangle$.

1.2 INVARIANCE LEMMA. *Let* A *and* B *be relational structures, of common signature* Γ, *isomorphic by* ϕ: $A \to B$. *Then for any* $S \in$ PROG(Σ), $\Sigma \subset \Gamma$, *and any input* $a_1,\ldots,a_n \in A$

$$\phi S(a_1,\ldots,a_n) \equiv_A S(\phi a_1,\ldots,\phi a_n).$$

The first-order assertion language L is based upon the syntactic vocabulary of PROG and is assumed to possess equality as well as the usual logical connectives and quantifiers. The semantics of the formulae and

sentences of L take their standard definition in model theory, see CHANG & KEISLER [7]. Corresponding to PROG(Σ) we take $L(\Sigma)$ to be the first-order sublanguage of L made from the constant, function and relation symbols appearing in the signature Σ.

Let $S \in \mathrm{PROG}(\Sigma)$ have named input variables $x = (x_1,\ldots,x_n)$ and output variable y. Let Γ be a signature extending Σ. If $\alpha = \alpha(x)$ and $\beta = \beta(y)$ are formulae of $L(\Gamma)$, having x and y as their free variables, then we can make a new kind of syntactic object, the so-called *asserted program* $\{\alpha\}S\{\beta\}$, the semantics of which is defined thus: for A a relational structure of signature Γ, the asserted program $\{\alpha\}S\{\beta\}$ is *valid* for A, written $A \models \{\alpha\}S\{\beta\}$, whenever $A \models \alpha(a)$ for $a \in A^n$ then either $S(a)\downarrow$ and $A \models \beta(S(a))$ or else $S(a)\uparrow$. In the obvious informal notation

$$A \models \{\alpha\}S\{\beta\} \text{ if, and only if, for all } a \in A^n,$$
$$A \models \alpha(a) \rightarrow [(S(a)\downarrow \wedge \beta S(a)) \vee S(a)\uparrow].$$

The following fact must be verified by the reader.

1.3 DEFINABILITY LEMMA. *Let* $S \in \mathrm{PROG}(\Sigma)$ *have input variables* $x = (x_1,\ldots,x_n)$ *and output variable* y. *Then for each* $\ell \in \omega$ *there exists a quantifier-free formula* $\mathrm{COMP}_{S,\ell}(x,y)$ *of* $L(\Sigma)$ *such that for each relational structure* A *whose signature contains* Σ, *and for all* $a \in A^n$, $b \in A$, $A \models \mathrm{COMP}_{S,\ell}(a,b)$ *if, and only if, the computation* $S(a)$ *terminates in* ℓ *steps or less and the output variable is valued at* b. *In symbols,*

$$A \models \mathrm{COMP}_{S,\ell}(a,b) \text{ if and only if, } |S(a)| \leq \ell \text{ and } S(a) = b.$$

Thus, by choosing a suitable polynomial $\mathrm{OUT}(x)$, *each individual terminating computation of* S *can be defined by a quantifier-free first-order formula of* $L(\Sigma)$ *of the form*

$$\mathrm{COMP}_{S,\ell}(x) \equiv \mathrm{COMP}_{S,\ell}(x,y) \wedge y = \mathrm{OUT}(x)$$

in the sense that for each A, *and all* $a \in A^n$

$$A \models COMP_{S,\ell}(a) \text{ } \textit{if, and only if,} \text{ } |S(a)| \leq \ell \text{ } \textit{and} \text{ } S(a) = OUT(a).$$

2. PROGRAMMING SYSTEMS AND DETERMINATENESS

Algorithms are written in a *definite* program formalism and are designed to compute functions over a *definite* data type semantics. The equation

algorithms = programs + data types

is a slogan implicit in this investigation in the sense that we use PROG(Σ) to fix the assignment, control and memory mechanisms available for the encoding of algorithms while the semantics of the data type primitives named in Σ are fixed by a class K of relational structures of signature Σ or some extension $\Gamma \supset \Sigma$.

A pair [K, PROG(Σ)] we call a *programming system*.

This first model of programming systems sees them as small scale program languages with a fixed range of data type primitives which are given an algebraic semantics. One can *imagine* that these programming systems are realised in some general purpose program language - by implementing their basic operators by functional procedures without side effects, for example. But *nothing is actually assumed of their data types' syntactic definition or abstract specification*, at least not at this point in our paper. Incidentally, the meaning of "algebraic semantics" here is exactly that in the current programming methodology literature: the semantics of data types are modelled by many-sorted algebras. (Our decision to work with "essentially" single-sorted program languages and their single-sorted interpretations is more a matter of notational convenience than technical necessity.)

For $S \in PROG(\Sigma)$ and K a class of relational structures of signature $\Gamma \supset \Sigma$, the *first-order partial correctness theory of* S *with respect to algebraic data type semantics* K is defined to be the set of preconditions and postconditions for asserted programs

$$PC_K(S) = \{(\alpha,\beta): \alpha,\beta \in L(\Gamma) \text{ and for each } A \in K, A \models \{\alpha\}S\{\beta\}\}.$$

The second clause of the definition we abbreviate $K \models \{\alpha\}S\{\beta\}$.

For $P,Q \in PROG(\Sigma)$, we say that P and Q are K-*equivalent* if for each $A \in K$, $P \equiv_A Q$. And this we abbreviate $P \equiv_K Q$.

The *first-order partial correctness theories are said to determine program equivalence, and therefore the input-output semantics, for the programming system* $[K, PROG(\Sigma)]$, if for every $P,Q \in PROG(\Sigma)$

$$PC_K(P) = PC_K(Q) \text{ implies } P \equiv_K Q.$$

This last property is what we take as the principal technical issue in formulating Floyd's thesis and, mathematically, this paper is given over to its study for various K. We shall refer to it as the *determinateness property* for the system $[K, PROG(\Sigma)]$.

DETERMINATENESS FOR A GENERAL PURPOSE PROGRAM LANGUAGE

The correctness theories determine the input-output semantics of the general purpose program language PROG *if they determine the input-output semantics of every specialised programming system it fathers*. We take the "sum" of the determinateness problems for all the $[K, PROG(\Sigma)]$ to be the determinateness problem for PROG.

In this formulation, the theoretical value of the determinateness property for PROG depends upon that of the formal model of a programming system. Since we are not yet assuming any conditions on our data type classes, the current determinateness property for PROG is enormously strong: the correctness theories are asked to determine program equivalence for some far-fetched examples of programming systems which PROG cannot implement. Nevertheless our first theorem, Theorem 3.1, will establish determinateness for PROG at this level of generality and in an apparently reasonable way. Indeed, the main objection to relying on the type of correctness theory used in that result is its surprising power.

It is more usual to see the semantical theory of PROG based on the pairs $[ALG(\Sigma), PROG(\Sigma)]$ where $ALG(\Sigma)$ is the species of all Σ-structures; this is the path taken by MEYER and HALPERN [28], for example. We do not take up this option because it misrepresents the relationship between the programming systems and the general language in which they are realised.

Moreover, we think it misrepresents the rôle played by data types in the proof theory that must be considered and so creates a misleading impression of the determinateness question for PROG. Of course, these issues can only be properly considered in the hindsight of Section 4.

SOME EXAMPLES

Consider a programming system $[K, PROG(\Sigma)]$ whose data type semantics has been defined uniquely up to isomorphism, the case of singleton classes $K = \{A\}$ containing all Σ-structures isomorphic to some representative structure A. This is one of the standard situations considered in the algebraic specification theory for data types where it is assumed that the semantics of a data type is modelled by *an algebra finitely generated by initial elements named in its signature*; such structures are called *minimal* because they contain no proper substructures. It is very easy to show

2.1 LEMMA. *Let* A *be a minimal structure of signature* Σ. *Then for any* $P,Q \in PROG(\Sigma)$

$$PC_A(P) = PC_A(Q) \text{ implies } P \equiv_A Q.$$

An immediate corollary of Lemma 2.1 is that the partial correctness theories determine program equivalence for PROG over the *standard model* of *arithmetic* $\underline{\underline{N}}$. $\underline{\underline{N}}$ we take to be the structure with domain the set of natural numbers ω, with the operations of successor, addition, and multiplication, with zero as distinguished constant, and with the ordering of ω as a basic relation. The signature of $\underline{\underline{N}}$ we write Σ_{arith}.

Lemma 2.1 also applies to the so called *prime rings* $\mathbb{Z}_n$ and $\mathbb{Z}$, and the *prime fields* $\mathbb{Z}_p$ and Q. The following proposition is designed to generate some equally simple counter-examples to determinateness without minimality.

2.2 LEMMA. *Let* K *be a class of* Σ*-structures satisfying these two properties: there is* $S \in PROG(\Sigma)$ *such that* (i) *for each* $A \in K$, S *computes an automorphism of* A, *and* (ii) *there exist* $A \in K$ *and* $a \in A$ *where* $S(a) \neq a$. *Then the first-order partial correctness theories fail to determine program equivalence for* $PROG(\Sigma)$ *over* K.

PROOF. Let P be the S hypothesised and let Q be a program for the identity map. Condition (ii) asserts that $P \not\equiv_K Q$ and we shall show $PC_K(P) = PC_K(Q)$. Assume for a contradiction that these sets do not coincide: let (α,β) lie in $PC_K(P)$ but not in $PC_K(Q)$, say. Using the facts that P,Q always compute total functions and that Q computes the identity we know that

$$\text{for each } A \in K,\ a \in A,\ A \models \alpha(a) \rightarrow \beta P(a), \text{ and}$$
$$\text{for some } B \in K,\ b \in B,\ B \not\models \alpha(b) \rightarrow \beta(b).$$

The second property implies $B \models \alpha(b) \wedge \neg\beta(b)$ and so we know that $B \models \alpha(b)$ and $B \not\models \beta(b)$, $B \models \beta P(b)$. But P computes an automorphism ϕ of B and since B is first-order we get a contradiction from the fact that $B \models \beta(x)$ if, and only if, $B \models \beta\phi(x)$ for any x.

The second case, where $(\alpha,\beta) \in PC_K(Q) - PC_K(P)$, is equally easy to check. Q.E.D.

Here are some examples where PROG(Σ) can be seen to loose determinateness on its straight-line programs.

2.3 FINITE FIELDS

Let F be a finite field of characteristic p. Then $\phi(x) = x^p$ is a field automorphism of F. If F is not $\mathbb{Z}_p$ then ϕ is not the identity. So take K to be any class of finite fields of characteristic p containing at least one $GF(p^n)$ for $n \neq 0,1$; in particular take $K = \{GF(p^n)\}$ with $n \neq 0,1$. (Remember that PROG is determined over $K = \{\mathbb{Z}_p\}$.) See PARIKH [31] in connection with this example.

2.4 LINEAR ALGEBRA

An *involution* $*$ of a (not necessarily commutative) ring R is an automorphism such that for all $r \in R$, $r^{**} = r$. Take K to be any class of rings with involution containing at least one R where the involution is not the identity. For example, let K contain the complex number field C with complex conjugation $a+ib \rightarrow a-ib$. Or let K contain the ring of 2×2 matrices over a field with the symplectic involution defined $\begin{pmatrix} a & b \\ c & d \end{pmatrix}^* = \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$.

The determinateness problem for programming systems of the form [A, PROG(Σ)] does *not* admit the clear cut solution suggested by Lemmas 2.1 and 2.2, however. Minimality is, indeed, an obvious condition to place on the interpretations which arise in the study of data type semantics and specifications. It stands between the simple idea that one wants names in the signature Σ of a specification (Σ,E) of a data structure A for initial values which generate the structure A, and the concept of initiality (say) which is a mathematical expression of *how* (Σ,E) specifies A: see ADJ [15]. But it is misleading when one considers the specification theory of data types in the wider context of programming systems as is done in Sections 5 and 6. There minimality disappears *because the semantics of the proof systems one needs depends upon* all *the models of its specifying axioms* ALG(Σ,E). The obvious example is ordinary arithmetic. It is trivial to give a concise algebraic axiomatisation of $\underline{\underline{N}}$, but the proof theory one needs for arithmetic computations on $\underline{\underline{N}}$ is *unavoidably* that based on Peano-like axioms as data type specifications and *all* their models as the data type semantics; Section 6 is devoted to this example.

TECHNICAL PRELIMINARIES

The definition of K-equivalence consists of two clauses: $P \equiv_K Q$ if, and only if, (1) for each $A \in K$ and all $a \in A^n$ such that $P(a)\downarrow$ and $Q(a)\downarrow$, $P(a) = Q(a)$ in A, and (2) for each $A \in K$ and all $a \in A^n$, $P(a)\downarrow$ if, and only if, $Q(a)\downarrow$. The first condition should naturally be called *weak* K-*equivalence* (cf. [24]), the second we call K-*convergence equivalence*. We denote these relations by $\equiv_{wK}$ and $\equiv_{cK}$ respectively. From these two notions we make two determinateness properties:

The partial correctness theories are said to *determine weak equivalence* for PROG(Σ) over K if for any $P,Q \in PROG(\Sigma)$

$$PC_K(P) = PC_K(Q) \text{ implies } P \equiv_{wK} Q.$$

The partial correctness theories are said to *determine convergence equivalence* for PROG(Σ) over K if for any $P,Q \in PROG(\Sigma)$

$$PC_K(P) = PC_K(Q) \text{ implies } P \equiv_{cK} Q.$$

2.3 LEMMA. *The program correctness theories determine program equivalence for* PROG(Σ) *over* K *if, and only if, they determine both weak and convergence equivalence over* K.

Let A be a structure and let $a_1,\ldots,a_n \in A$. Then adjoining these elements $a_1,\ldots,a_n$ to A as distinguished constants makes a new structure denoted $(A,a_1,\ldots,a_n)$.

A *closed program* S is one without any uninitialised variables.

2.4 CONVERGENCY LEMMA. *Let* K *be a class of structures of common signature* Γ *and let* $\Sigma \subset \Gamma$. *The following condition is sufficient for the partial correctness theories to determine convergence equivalence for* PROG(Σ) *over* K.

For any finite extension of Σ *by constants,* $\Sigma(\underline{\underline{c}}) = \Sigma \cup \{\underline{\underline{c}}_1,\ldots,\underline{\underline{c}}_n\}$, *and for any closed program* S *over* $\Sigma(\underline{\underline{c}})$, *if* S *diverges on some algebra in* K *then there is a sentence* Θ, *first-order over* $\Sigma(\underline{\underline{c}})$, *which is satisfied in some algebra in* K *and such that for any* $A \in K$ *and* $a_1,\ldots,a_n \in A$, $(A,a_1,\ldots,a_n) \models \Theta \rightarrow S\!\uparrow$.

PROOF. Assuming the condition holds we are to prove that for any $P,Q \in$ PROG(Σ)

$$PC_K(P) = PC_K(Q) \text{ implies } P \equiv_{cK} Q$$

So, contrapositively, suppose that P,Q are Σ-programs for which there exists $A \in K$ and $a = (a_1,\ldots,a_n) \in A^n$ where P(a) converges but Q(a) diverges (say). Let $|P(a)| = t$ and define a new program abbreviated by

$$S(x) \equiv \underline{\text{if}}\ COMP_{P,t}(x)\ \underline{\text{then}}\ Q(x)\ \underline{\text{else}}\ \text{STOP}\ \underline{\text{fi}}$$

Notice that S has signature Σ and, moreover, it does not require programming features beyond those assumed for P,Q. (This is because a straight-line program over Σ can be written to decide $COMP_{P,t}(x)$.)

Adding the new constants $\underline{\underline{c}}_1,\ldots,\underline{\underline{c}}_n$ to Σ we create the closed program $S(\underline{\underline{c}})$ over $\Sigma(\underline{\underline{c}})$ by replacing input variable x_i of S with constant $\underline{\underline{c}}_i$, $1 \leq i \leq n$. By hypothesis, $(A,a_1,\ldots,a_n) \not\models S(\underline{\underline{c}})\!\uparrow$. And applying the condition we get a sentence Θ which is first order $\Sigma(\underline{\underline{c}})$ and K-consistent and which

implies the divergence of $S(\underline{\underline{c}})$ throughout K. Let $\Theta_0(x)$ be Θ with each constant $\underline{\underline{c}}_i$ replaced by variable x_i, $1 \leq i \leq n$.

We claim: $(\Theta_0(x), \underline{\text{false}}) \in PC_K(Q) - PC_K(P)$.

The pair cannot lie in $PC_K(P)$ because whenever $\Theta_0(x)$ is true $S(x)$ diverges and, by definition, $P(x)$ converges and $\{\Theta_0\}P\{\underline{\text{false}}\}$ is not true. On the other hand $\{\Theta_0\}Q\{\underline{\text{false}}\}$ is valid for K because $\Theta_0(x)$ is true implies $Q(x)$ must diverge. Q.E.D.

3. DETERMINATENESS VIA EXTENDED SEMANTICS

Consider a typical programming system [K, PROG(Σ)] wherein K is *any* class of Σ-structures. Our first project is to show that a conservative and uniformly definable extension of the assertion language and its semantics enables the partial correctness theories to determine program equivalence at this level of generality.

Let Σ_{arith} = {0, SUCC, ADD, MULT, ORDER} be the signature of arithmetic assumed disjoint from Σ. A structure B, of signature $\Sigma \cup \Sigma_{arith}$, is a *(formal) arithmetical expansion* of a Σ-structure A if the Σ-reduct of B is isomorphic to A.

Let AE(K) denote the class of all arithmetical expansions of all algebras in K.

3.1 BASIC EXTENSION THEOREM. *Let* K *be any class of* Σ*-structures. Then for any* $P,Q \in PROG(\Sigma)$

$$PC_{AE(K)}(P) = PC_{AE(K)}(Q) \text{ implies } P \equiv_K Q$$

Let us first consider the positive aspects of Theorem 3.1 and postpone our reservations until after its proof.

This theorem is, indeed, a striking result in faviour of Floyd's thesis, especially when one sets the weakness of first-order assertion languages against the strength of PROG and the generality of K. As far as the input-output semantics of PROG(Σ) is concerned, the two programming systems are identical: each $S \in PROG(\Sigma)$ is interpreted by the same class of Σ-structures and over each such structure S computes the same function. If K is specified

by first-order axioms over Σ, or if K is specified by algebraic axioms in conjunction with initial algebra semantics (ADJ [15]), then, in both cases, *precisely* these axioms over $\Sigma \cup \Sigma_{arith}$ uniquely characterise AE(K). Indeed, Theorem 3.1 is "best possible" in the sense that a result for arbitrary K cannot avoid the use of hidden functions in the assertion language; we have seen this in Lemma 2.2. And, in any case, it is known in the algebraic theory of data types that to specify all the data types one wants it is necessary (and sufficient) to use hidden functions from the language set aside for the purpose.

In MEYER and HALPERN [28] the role of our Theorem 3.1 is played by their Theorem 4.1.

PROOF OF THE BASIC EXTENSION THEOREM

Let $P,Q \in PROG(\Sigma)$. Observe that $P \equiv_K Q$ if, and only if, $P \equiv_{AE(K)} Q$: we shall prove that

$$PC_{AE(K)}(P) = PC_{AE(K)}(Q) \text{ implies } P \equiv_{AE(K)} Q.$$

First consider convergence equivalence. By the Convergency Lemma 2.4, it is sufficient to examine closed programs over finite extensions by constants of Σ. Let S be a closed program over $\Sigma(\underline{\underline{c}})$ where $\underline{\underline{c}} = (\underline{\underline{c}}_1,\ldots,\underline{\underline{c}}_n)$. Suppose $A \in AE(K)$ and that for $a_1,\ldots,a_n \in A$ we have $(A,a_1,\ldots,a_n) \models S\uparrow$. We shall construct a sentence Θ first-order over $\Sigma(\underline{\underline{c}}) \cup \Sigma_{arith}$ which is satisfied on $(A,a_1,\ldots,a_n)$ and such that for each $B \in AE(K)$ and each $b_1,\ldots,b_n \in B$

$$(B,b_1,\ldots,b_n) \models \Theta \rightarrow S\uparrow.$$

Now if A is finite then the Σ-substructure of A generated by $a_1,\ldots,a_n$ is finite and can be defined, up to isomorphism, by a first-order sentence $\theta(\underline{\underline{c}})$ over $\Sigma(\underline{\underline{c}})$. This $\theta(\underline{\underline{c}})$ is trivially satisfied on $(A,a_1,\ldots,a_n)$ and, using the Locality Lemma 1.1 and the Invariance Lemma 1.2, it is easy to show for any $B \in AE(K)$ and any $b_1,\ldots,b_n \in B$ that if $(B,b_1,\ldots,b_n) \models \theta(\underline{\underline{c}})$ then $(B,b_1,\ldots,b_n) \models S\uparrow$. Therefore, we may take $\theta(\underline{\underline{c}})$ to be Θ. (Notice no hidden

functions were required here.)

Assume A is infinite. We define a unary formula N, first-order over Σ_{arith} by

$$N(x) \equiv \exists y.SUCC(y) = x$$

Without loss of generality it can be further assumed of A that

(i) $A_N = \{a \in A: A \models N(a)\}$ is a $\Sigma \cup \Sigma_{arith}$ substructure of A and contains $\langle a_1,\ldots,a_n\rangle$; in a nutshell, A_N is a $\Sigma(\underline{c}) \cup \Sigma_{arith}$ substructure of $(A,a_1,\ldots,a_n)$.

(ii) The reduct $A_N|_{\Sigma_{arith}}$ is isomorphic to $\underline{\underline{N}}$.

This transformation is easy to arrange. First fix that $\langle a_1,\ldots,a_n\rangle$ (is or) lies in a countably infinite Σ-substructure X of A. Then define O, SUCC, ADD, MULT on X and A-X such that $A_N = X$.

Since $(A,a_1,\ldots,a_n) \models S\uparrow$, for each $t \in \omega$, $A \models \neg COMP_{S,t}(\underline{c})$. And the next step is to formalise an arithmetisation of $COMP_{S,t}(\underline{c})$ in the first-order language of $\Sigma(\underline{c}) \cup \Sigma_{arith}$. For $t \in \omega$ we denote by $\underline{t}$ the term $SUCC^t(0)$ over Σ_{arith}.

3.2 REPRESENTATION LEMMA. *Let Γ be any signature and let $\{\theta_t(x): t \in \omega\}$ be a recursively enumerable sequence of open formulae of $L(\Gamma)$ with variables $x = (x_1,\ldots,x_n)$. Then there exists a sentence Ψ and a formula $\theta(y,x)$ in the first-order language of $\Gamma \cup \Sigma_{arith}$ such that*

(i) *Ψ is true in any $\Gamma \cup \Sigma_{arith}$ structure A in which $A_N = \{a \in A: \exists y.SUCC(y) = a\}$ is a $\Gamma \cup \Sigma_{arith}$ substructure of A and $A_N|_{\Sigma_{arith}} \cong \underline{\underline{N}}$;*

(ii) *for each $t \in \omega$, $\Psi \vdash N(x_1) \wedge\ldots\wedge N(x_n) \rightarrow [\theta_t(x) \leftrightarrow \theta(\underline{t},x)]$.*

We do not stop to prove this lemma as its argument is a rather straightforward adaptation of the proof of the representability of the recursive functions in arithemtic.

Applying the Representation Lemma 3.2 with $\Gamma = \Sigma(\underline{c})$ and $\theta_t = \neg COMP_{S,t}(\underline{c})$ we choose appropriate Ψ and $\theta(y) = \theta(y,\underline{c})$, first-order over $\Sigma(\underline{c}) \cup \Sigma_{arith}$. By our choice of A, we can get from the lemma that $(A,a_1,\ldots,a_n) \models \Psi$ and so

$$(A,a_1,\ldots,a_n) \models \theta(\underline{t},\underline{c}) \leftrightarrow \neg COMP_{S,t}(\underline{c})$$

Therefore, $(A,a_1,\ldots,a_n) \models \theta(\underline{t},\underline{c})$ for every $t \in \omega$.

The sentence we require is

$$\Theta \equiv \Psi \wedge \forall y.[N(y) \rightarrow \theta(y,\underline{c})]$$

We now verify the local condition of the Convergency Lemma 2.4. Θ is clearly first-order over $\Sigma(\underline{c}) \cup \Sigma_{arith}$ and is consistent by its construction. Suppose $B \in AE(K)$ and $b_1,\ldots,b_n \in B$ are such that $(B,b_1,\ldots,b_n) \models \Theta$. Then $(B,b_1,\ldots,b_n) \models \theta(\underline{t},\underline{c})$ for all t and, using

$$\Psi \vdash \theta(\underline{t},\underline{c}) \leftrightarrow \neg COMP_{S,t}(\underline{c})$$

we may deduce that

$$(B,b_1,\ldots,b_n) \models \bigwedge_{t\in\omega} \neg COMP_{S,t}(\underline{c})$$

which means the program diverges.

We now consider how the partial correctness theories determine weak equivalence for Σ-programs over AE(K).

Let $P,Q \in PROG(\Sigma)$ and let $A \in AE(K)$. Suppose that for $a = (a_1,\ldots,a_n) \in A^n$, P(a) and Q(a) converge to distinct values. Taking $|P(a)| = k$ and $|Q(a)| = \ell$ we define the difference formula

$$DIFF(x) \equiv COMP_{P,k}(x) \wedge COMP_{Q,\ell}(x) \wedge OUT_{P,k}(x) \neq OUT_{Q,\ell}(x).$$

It is now straightforward to separate the correctness theories of P and Q with the pair (α,β) defined by

$$\alpha(x) \equiv DIFF(x) \wedge \bigwedge_{i=1}^{n} [S^{f(i)}(0) = x_i]$$

$$\beta(y) \equiv y = OUT_{P,k}(S^{f(0)}(0),\ldots,S^{f(n)}(0))$$

wherein $f(i) = (\mu j)[a_j = a_i]$. We leave to the reader the task of verifying

$(\alpha,\beta) \in PC_{AE(K)}(P)$ but $(\alpha,\beta) \notin PC_{AE(K)}(Q)$. Q.E.D.

Although we accept Theorem 3.1 as a respectable theoretical statement about Floyd's thesis we also see it as a reference point which dictates a refinement of the analysis to be in order. This refinement we *organise* around the question *Under what circumstances can the hidden functions be eliminated?* It is the proof of the theorem itself which forces this opinion. The argument rests on the remarkable definability properties of the recursive functions on the natural numbers: rather than internalising or imitating this number-theoretic machinery within the semantics of the programming system that is given, we have simply expanded the semantics to make use of it. (If we had applied this technique solely to the [ALG(Σ), PROG(Σ)] then we would have obscured its power and, for that matter, its technical structure.) Undermining the satisfactory features of Basic Extension Theorem 3.1, documented prior to its proof, is the feeling that stronger, and still fairly general, results are possible and that, in particular, these results would be more illuminating even if they are no more conclusive as far as the viability of Floyd's principle is concerned. Certainly, Theorem 3.1 seems to say as much about the power of the recursion-theoretic equipment as it does about the semantical problems involved.

4. ELIMINATING HIDDEN FUNCTIONS

This section is devoted to proving the following theorem, the most difficult to be found in our paper.

4.1 THEOREM. *Let Σ be any signature except one containing exclusively unary relations and at most one unary function symbol. Then for any* $P,Q \in PROG(\Sigma)$

$$PC_{ALG(\Sigma)}(P) = PC_{ALG(\Sigma)}(Q) \text{ implies } P \equiv_{ALG(\Sigma)} Q.$$

Mathematically, Theorem 4.1 represents the fate of the plausible observation that one has only to internalise the arithmetic mechanisms, seen in the proof of Theorem 3.1, to rid oneself of the hidden functions: *it can almost be done, but only when there are no requirements placed on the data type semantics*. To this we add the conjecture:

4.2 CONJECTURE. *For those signatures of the kind explicitly ruled out in the hypothesis of Theorem* 4.1 *the conclusion of that theorem is false.*

Those readers who prefer to conceive of the semantical theory of PROG as being determined by programming systems [ALG(Σ), PROG(Σ)] should attach quite some weight to Theorem 4.1 and to the open problem represented by Conjecture 4.2. (Remember: any program naming only the constant zero, the successor function, and some unary boolean conditions for an arithmetical computation is left uncovered by Theorem 4.1.) Although the theorem has less bearing on Floyd's principle in the context of our own analysis, its proof is of great technical interest when contrasted with the proof of Theorem 3.1.

Theorem 4.1 appears, in a slightly weaker form, as Theorem 7.2 in MEYER and HALPERN [28].

PROOF OF THEOREM 4.1

The plan of the argument is this. We begin by proving determinacy for weak equivalence. The proof of determinacy for convergency is based upon the Convergency Lemma 2.4 and it divides into a singular case, in which Σ contains one unary function symbol and some constants, and the usual one of those signatures remaining. The argument for the usual case is, indeed, involved and we take it next leaving the singular case as a loose end to conclude the section.

4.3 LEMMA. *Let* Σ *be a signature containing at least one function symbol. Then the partial correctness theories determine weak program equivalence on* ALG(Σ).

PROOF. Suppose there is a Σ-structure A and $a = (a_1,\ldots,a_n) \in A^n$ such that P(a) and Q(a) converge to distinct values. Let $|P(a)| = k$ and $|Q(a)| = \ell$ and define the difference formula

$$DIFF(x) \equiv COMP_{P,k}(x) \wedge COMP_{Q,\ell}(x) \wedge OUT_{P,k}(x) \neq OUT_{Q,\ell}(x),$$

where $x = (x_1,\ldots,x_n)$. It is sufficient to make a first-order definition over Σ of some $x = (x_1,\ldots,x_n)$ for which DIFF(x) holds throughout ALG(Σ) and to show consistency. We will construct unary formulae $\phi_1,\ldots,\phi_n$ such that

$$\Phi(x) \equiv \bigwedge_{i=1}^{n}\phi_i(x_i) \wedge \exists x_1,\ldots,x_n.\ \bigwedge_{i=1}^{n}\phi_i(x_i) \wedge \mathrm{DIFF}(x)$$

is consistent. This done, it is easy to check that the pair (α,β) defined by

$$\alpha(x) \equiv \Phi(x)\,,\quad \beta(x) \equiv \exists x_1,\ldots,x_n.\ \bigwedge_{i=1}^{n}\phi_i(x_i) \wedge y = \mathrm{OUT}_{P,k}(x)$$

lies in $\mathrm{PC}_\Sigma(P) = \mathrm{PC}_{\mathrm{ALG}(\Sigma)}(P)$ but not in $\mathrm{PC}_\Sigma(Q) = \mathrm{PC}_{\mathrm{ALG}(\Sigma)}(Q)$.

We will first construct a Σ-structure B which is to witness the consistency of Φ. Let f be a k-ary operation of A. The structure B is simply A with this operation f redefined along its diagonal.

The computations P(a) and Q(a) take place within the subsystem $\langle a\rangle$ of A generated by $a = (a_1,\ldots,a_n) \in A^n$ (Locality Lemma 1.1). Let $X \subset \langle a\rangle$ be the set of all elements appearing in either of these computations. Now choose some $u \in A-X$ and $\frac{1}{2}n(n+1)$ distinct elements $\{b_{ij}: 1 \le i < j \le n\}$ from $A - X\cup\{u\}$. Define $g: A^k \to A$ by

$$\begin{aligned}
g(y_1,\ldots,y_k) &= f(y_1,\ldots,y_k) && \text{if } y_i \neq y_j \text{ for some } 1 \le i,j \le n\\
&= a_j && \text{if } y = b_{ij}\\
&= f(y,\ldots,y) && \text{if } y \in X \text{ and } f(y,\ldots,y) \in X\\
&= u && \text{if } y \in X \text{ and } f(y,\ldots,y) \notin X\\
&= u && \text{if } y = u\\
&= y && \text{otherwise.}
\end{aligned}$$

Replacing f by g in A makes B. We now leave to the reader the task of verifying what remains of the proof on taking

$$\phi_i(x_i) \equiv \exists z_1,\ldots,z_i\Big[\bigwedge_{1\le s<t\le i} z_s \neq z_t \wedge \bigwedge_{1\le j\le i} \neg\exists z.f(z,\ldots,z) = z_j \wedge \bigwedge_{1\le j\le i} f(z_j,\ldots,z_j) = x_i\Big].$$

Q.E.D.

4.4 LEMMA FOR THE USUAL CASES. *Let* Σ *be (a finite extension by constants of) a signature containing at least two functions or at least a function of arity greater than one or a unary function and a relation of arity greater than one. Then for any closed program* S *over* Σ, *if for some* $A \in ALG(\Sigma)$ $A \models S\uparrow$ *then there is a sentence* ϕ, *first-order over* Σ, *which is consistent with* $ALG(\Sigma)$ *and such that* $ALG(\Sigma) \models \phi \rightarrow S\uparrow$.

PROOF. The proof of the lemma is based upon the argument for convergency in the proof of the Basic Extension Theorem 3.1 where arithmetic syntax is adjoined to obtain K-consistent formulae implying convergence for closed programs throughout K. The pleasant feature there is that the arithmetic required in no way interferes with the computations considered since the programs make reference only to operations prescribed for K. Here, however, we are to make available comparably strong, but *internal*, mechanisms. Our techniques to do this have some set theory in the rôle of arithmetic and will make full use of the freedom to manoeuvre, model-theoretically, characteristic of $ALG(\Sigma)$.

We will formulate the machinery in general terms using a 2-sorted first-order language destined to be interpreted in $L(\Sigma)$ with the result that the bulk of the proof will then rest on Lemma 4.5 about its specification and its interpretation. After Lemma 4.5 we have to show that each signature Σ admits an appropriate interpretation, a task which depends on the composition of the signature and cannot be made uniform.

Let Γ be any single-sorted signature. This we expand to a 2-sorted signature Γ_2 by adding to Γ a new sort called SET, and renaming by DOM the original (implicit) sort of Γ, together with the binary relations $\in$ and CODE of sorts SET $\times$ SET and SET $\times$ DOM respectively. Given $A \in ALG(\Gamma_2)$ we denote by $A|_{DOM}$ the Γ-reduct of A and by $A|_{SET}$ the $\{SET,\in\}$-reduct of A.

The first-order language $L(\Gamma_2)$ over Γ_2 has two kinds of variables

$$x_1^D, x_2^D, \ldots \quad \text{and} \quad x_1^S, x_2^S, \ldots$$

ranging over sorts DOM and SET respectively although we drop the superscripts whenever confusion seems unlikely. We assume, for brevity, the languages $L(\Gamma)$ and $L(\Gamma_2)$ use only the connectives $\neg$, $\vee$ and the quantifier $\exists$.

Now an interpretation of $L(\Gamma_2)$ in $L(\Gamma)$ is determined as soon as formulae of $L(\Gamma)$ are chosen to define predicates for each sort and to define the relations ϵ and CODE. Suppose we are given four formulae of $L(\Gamma)$, say the list $I = \{Q_D(x), Q_S(x), Q_\epsilon(x,y), Q_c(x,y)\}$. This list I determines an interpretation $H^I: L(\Gamma_2) \to L(\Gamma)$ in an obvious way:

$$
\begin{aligned}
&H^I(x_i^D) = x_{2i}\\
&H^I(x_i^S) = x_{2i+1}\\
&H^I(f(t_1,\ldots,t_k)) = f(H^I(t_1),\ldots,H^I(t_k))\\
&H^I(x_i^S \in x_j^S) = Q_\epsilon(H^I(x_i^S),H^I(x_j^S))\\
&H^I(\mathrm{CODE}(x_i^S,t)) = Q_c(H^I(x_i^S),H^I(t))\\
&H^I(\Phi \vee \Psi) = H^I(\Phi) \vee H^I(\Psi)\\
&H^I(\neg\Phi) = \neg H^I(\Phi)\\
&H^I(\exists x_i^D.\Phi) = \exists x_{2i}.(Q_D(x_{2i}) \wedge H^I(\Phi))\\
&H^I(\exists x_i^S.\Phi) = \exists x_{2i+1}.(Q_S(x_{2i+1}) \wedge H^I(\Phi))
\end{aligned}
$$

where f is a k-ary operation of Γ, $t, t_1, \ldots, t_k$ are Γ-terms; and Φ, Ψ are formulae of $L(\Gamma_2)$.

We are able to prove Lemma 4.4 for precisely those signatures Σ which admit interpretations H^I satisfying the hypothesis of the following general lemma.

4.5 LEMMA. *Let Θ be a sentence of $L(\Gamma)$ and let H^I be an interpretation of $L(\Gamma_2)$ into $L(\Gamma)$ which together satisfy these two conditions:*

(1) *Given any closed program S over Γ which diverges on some Γ-structure there exists a Γ-structure A where $A \models \Theta$ and $A \models S\uparrow$.*

(2) *For any sentence Ψ of $L(\Gamma_2)$ whenever $\Theta \wedge \Psi$ is consistent with respect to $ALG(\Gamma_2)$ then $H^I(\Theta \wedge \Psi)$ is consistent with respect to $ALG(\Gamma)$.*

Then given any closed program S over Γ which diverges somewhere in $ALG(\Gamma)$ there exists a sentence ϕ of $L(\Gamma)$ which is consistent with respect to $ALG(\Gamma)$ and $ALG(\Gamma) \models \phi \to S\uparrow$.

PROOF. Let S be a closed program over Γ diverging somewhere in ALG(Γ). By condition (1) we can choose an $A \in ALG(\Gamma)$ on which S diverges and $A \models \Theta$. Let $M(\Theta)$ be the subclass of ALG(Γ) composed of those algebras satisfying Θ. Using the arguments for convergence in the proof of the Basic Extension Theorem 3.1, we may find a sentence ϕ_0, first-order over $\Gamma \cup \Sigma_{arith}$, such that

$$ALG(\Gamma \cup \Sigma_{arith}) \cap M(\Theta) \models \phi_0 \rightarrow S\uparrow$$

and ϕ_0 is there consistent.

From ϕ_0 we shall construct a sentence Φ of $L(\Gamma_2)$ such that

(i) $\Theta \wedge \Phi$ is $ALG(\Gamma_2)$ consistent and $ALG(\Gamma_2) \models \Phi \rightarrow S\uparrow$.

Therefore, $H^I(\Theta \wedge \Phi)$ is ALG(Γ) consistent, by condition (2), and we may take $\phi = H^I(\Theta \wedge \Phi)$ and prove

(ii) $ALG(\Gamma) \models \phi \rightarrow S\uparrow$

which completes the argument for the lemma.

First of all let us prove this latter statement (ii) assuming Φ to have been constructed and that it satisfies statement (i).

Suppose $A \in ALG(\Gamma)$ satisfies $\phi = H^I(\Theta \wedge \Phi)$. We extend A to a Γ_2-structure B by adding $\{a \in A: A \models Q_S(a)\}$ as a set-theoretic domain and defining $\in$ and CODE for B by $Q_\in$ and Q_c. Now for every sentence $\delta \in L(\Gamma_2)$

$$B \models \delta \text{ if, and only if, } A \models H^I(\delta).$$

Therefore, $B \models \Theta \wedge \Phi$ and, by condition (i), $B \models S\uparrow$ which means $B \models$ $B \models \bigwedge_{k \in \omega} \neg COMP_{S,k}$. Applying H^I to this formula, and using the fact that $H^I(\neg COMP_{S,k}) = \neg COMP_{S,k}$, we deduce $A \models S\uparrow$.

We now construct Φ from ϕ_0 and prove statement (i). Here is a technical lemma whose proof is a tedious exercise in axiomatic set theory which we take the liberty of omitting.

4.6. LEMMA. *Let Δ be a finite signature and let δ be a sentence of $L(\Delta)$. Then there is a sentence p and a formula q(x) of the first-order language of Zermelo-Fraenkel set theory L(ZF) such that*

(a) $ZF \vdash p$.

(b) *If* $B \models p$ *then for* $b \in B$, $B \models q(b)$ *if, and only if,* b *is a* Δ*-structure which satisfies the sentence* δ.

This lemma we apply to $\Delta = \Gamma \cup \Sigma_{arith}$ and $\delta = \phi_0 \wedge \Theta$ to obtain appropriate p and q(x). Let Ψ be a sentence, first-order over Γ_2, which expresses the following property of a Γ_2-structure B:

"If p and $\exists x.q(x)$ are true of B then for some $b \in B|_{SET}$, q(b) holds and CODE restricted to $\{b' \in B|_{SET}: b' \in b\} \times B|_{DOM}$ is the graph of a Γ-isomorphism $b \to B|_{DOM}$."

We set $\Phi \equiv p \wedge \exists x.q(x) \wedge \Psi$ and aim to show this Φ is $ALG(\Gamma_2)$-consistent and that $ALG(\Gamma_2) \models \Phi \to S\uparrow$.

Consider consistency. We seek a Γ_2-structure C satisfying Φ. For that part of C of sort DOM we choose any $B \in ALG(\Gamma \cup \Sigma_{arith})$ such that $B \models \phi_0$. For the set-theoretic part of C we take any model of Zermelo-Fraenkel set theory containing an element b which is in fact a $\Gamma \cup \Sigma_{arith}$-structure isomorphic to B. And, to complete the construction of C, we define CODE as the graph of any function which restricted to b is a Γ-isomorphism $b \to B$. It is easy to verify $C \models \Phi$.

Consider divergence. Let A be any Γ_2-structure with $A \models \Theta \wedge \Phi$. Choose $a \in A|_{SET}$ such that $A \models q(a)$. As $A \models p$ we know a is a $\Gamma \cup \Sigma_{arith}$-structure which satisfies ϕ_0. As $ALG(\Gamma \cup \Sigma_{arith}) \cap M(\Theta) \models \phi_0 \to S\uparrow$ we know $a \models S\uparrow$. But CODE, under these hypotheses, represents an isomorphism $a \to A|_{DOM}$ and, therefore, S diverges on A.

This concludes the proof of Lemma 4.5.

To complete the argument for the usual cases is a matter of defining interpretations H^I for the various signatures and proving true of them the two hypotheses of Lemma 4.5. We give two representative cases:

(i) Σ contains one binary function f and a constant c.

Here take
$$\Theta \equiv \exists x,y.[x \neq y \wedge \forall x.\exists y(f(y,y) = x)]$$
$$Q_D(x) \equiv \exists y.f(y,y) = x$$
$$Q_S(x) \equiv \neg\exists y.f(y,y) = x$$
$$Q_\in(x,y) \equiv x \neq y \wedge f(x,y) = c$$
$$Q_c(x,y) \equiv x = f(y,y)$$

(ii) Σ contains two unary functions f,g.

Here take $\Theta \equiv \forall x \exists y.f(y) = x$

Let $Q(x) \equiv \neg\exists y.f(y) = x$ and define

$$Q_S(x) \equiv \exists y.(Q(y) \wedge f(y) = x)$$
$$Q_D(x) \equiv \neg Q(x) \wedge \neg Q_S(x)$$
$$Q_\in(x,y) \equiv \exists z.(Q(z) \wedge f(z) = x \wedge g(z) = y)$$
$$Q_c(x,y) \equiv f(x) = y.$$

Q.E.D.

4.6 LEMMA FOR THE SINGULAR CASE. *Let Σ be (a finite extension by constants of) a signature containing exactly one unary function and let K be any class of Σ-algebras which is closed under taking subalgebras. Then for any closed program S over Σ, if for some $A \in K$ $A \models S\uparrow$ then there is a sentence ϕ, first-order over Σ, which is consistent with K and such that $K \models \phi \to S\uparrow$.*

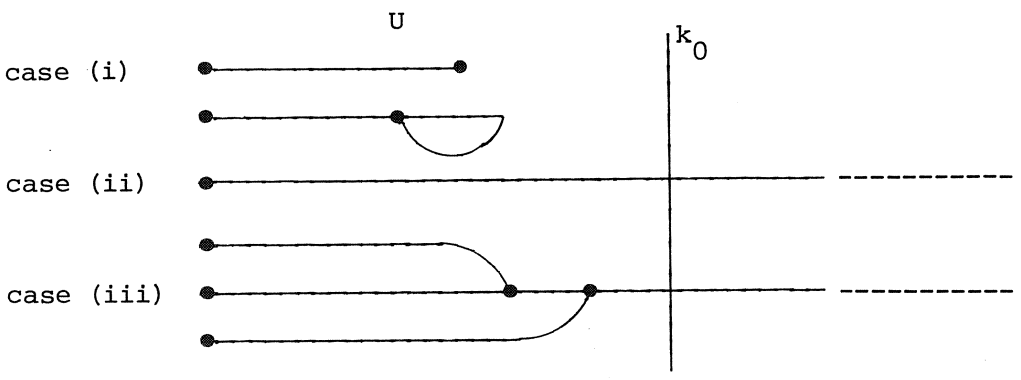
PROOF. Assume $S = S(\underline{c})$ is a closed program over Σ involving $\underline{c} = (\underline{c}_1,\ldots,\underline{c}_n)$ and that $A \in K$ is such that $A \models S\uparrow$ when $\underline{c}$ is interpreted by $a = (a_1,\ldots,a_n)$. We make a special decomposition of the subalgebra $\langle a_1,\ldots,a_n\rangle$ of A. For f the unary operation of A and for *any* $a \in A$ and $k \in \omega$ define

$$orb_k(f,a) = \{b \in A: \exists i < k,\ f^i(a) = b\}$$

and then

$$orb(f,a) = \bigcup_{k\in\omega} orb_k(f,a).$$

Thus, $\langle a_1,\ldots,a_n\rangle = orb(f,a_1) \cup \ldots \cup orb(f,a_n)$. There arises just a few possible types of *orbit* in this decomposition of interest to us, illustrated in the figure below: (i) $orb(f,a_i)$ is finite; (ii) $orb(f,a_i)$ is infinite and meets no other orbit; (iii) $orb(f,a_i)$ is infinite but intersects some $orb(f,a_j)$. In this third case notice that if $b \in orb(f,a_i) \cap orb(f,a_j)$ then $f^r(a_i) = b = f^s(a_j)$ for some r,s and hence for all k, $f^{r+k}(a_i) = f^{s+k}(a_j)$.



Choose k_0 so large as to bound the cardinalities of the finite orbits and the finite parts of intersecting orbits which remain distinct; set

$$U = \bigcup_{\substack{k\leq k_0 \\ i\leq n}} orb_k(f,a_i)$$

We aim to represent this subalgebra structure in a first-order sentence over Σ.

Let U be defined by the formula

$$U(x) = \bigvee_{\substack{i\leq k_0 \\ j\leq n}} x = f^i(\underline{c}_j).$$

Let R define all equalities and inequalities in U in this way: set

$$T(i,j,p,q) = \begin{cases} f^i(\underline{c}_p) = f^j(\underline{c}_q) & \text{if } f^i(a_p) = f^j(a_q) \text{ in A} \\ f^i(\underline{c}_p) \neq f^j(\underline{c}_q) & \text{otherwise.} \end{cases}$$

Then

$$R = \bigwedge_{\substack{i,j\leq k_0 \\ i,q\leq n}} T(i,j,p,q)$$

Let

$$W = (\forall x)[\neg U(x) \rightarrow f(x)\neq x \wedge \neg U(f(x)) \wedge \forall y,z.[f(y)=f(z)=x\rightarrow y=z]].$$

And choosing those $a_{\lambda_1},\ldots,a_{\lambda_t}$ such that for all $b \in U$, $f(b) \neq a_{\lambda_i}$ $(1\leq i \leq t)$ we define

$$V = \bigwedge_{i\leq t} \forall x.[f(x) \neq \underline{c}_{\lambda_i}].$$

Let $\phi = R \wedge W \wedge V$. We claim ϕ to be K-consistent and that $K \models \phi \rightarrow S\uparrow$. The consistency of ϕ follows from its construction from $\langle a_1,\ldots,a_n\rangle$ and the hypothesis that subalgebras of K-algebras are again K-algebras. To obtain $K \models \phi \rightarrow S\uparrow$ one proceeds as follows. Let $B \in K$ and $B \models \phi$. Let B' be the subalgebra of B generated by the elements of B named by the constants in S. One can now show that $B' \models \phi$ implies B' is isomorphic to $\langle a_1,\ldots,a_n\rangle$.

Therefore, S diverges on B' by Invariance Lemma 1.2, and so B $\models$ S↑ by Locality Lemma 1.1. The proof of the isomorphism we leave to the reader. Q.E.D.

Given some sympathy for our conception of a programming system, the methods used to internalise the hidden operators of Theorem 3.1 which go into the proof of Theorem 4.1 can be seen as an abuse of the semantical components of Floyd's thesis. Underlining our reservations about modelling the semantics of PROG through [ALG(Σ), PROG(Σ)] is the fact that this view of a general programming language sees these techniques as quite acceptable.

5. FLOYD'S PRINCIPLE AND PROGRAMMING SYSTEMS WITH SPECIFICATIONS

In Sections 2,3 and 4 we have achieved our first objective of providing a fairly thorough account of the determinateness problem for a liberal model of a programming system and, by extension, for a liberal formulation of the determinateness problem for a general purpose programming language. Certainly, with our current definitions, we have exhausted the implications of the determinateness problem for Floyd's thesis. We are now to start on a second analysis, one which forgets about general programming languages (and so parts company with MEYER and HALPERN [28]) and is carefully tailored to specialised programming systems. Of course, we know from our Basic Extension Theorem 3.1 that some hidden functions in the assertion language and an expansion of the data type semantics of the programming system will settle the problem at once. Our objective here is to think through the issues without recourse to the remarkable, but extrinsic, powers of recursion-theoretic definability theory. Instead, we will take as a guide certain reasonable assumptions about modelling a programming system with a limited field of application. Our main idea is that *the data types of such a programming system must be syntactically specified and that its specification* (Σ,E) *has an essential role to play in the construction of any proof theory for partial correctness in the system.* This new parameter, the specification, allows us to search for new information about Floyd's principle through more delicate mathematical experiments in the style of algorithmic logic.

In this penultimate section, we present a new technical exegesis of the determinateness problem which is designed to overcome the hasty counter-

examples associated with Lemma 2.2. After this, we encorporate specifications into the models of programming systems and prove some basic results about determinateness in these new systems. In the last subsection, we carefully analyse several programming systems made to handle arithmetical computations.

THE DETERMINATENESS PROBLEM FOR PROGRAMMING SYSTEMS REVISITED

We circumvent Lemma 2.2 with a new definition of partial correctness theories taken from MANNA [24, pp.164].

Let $S \in PROG(\Sigma)$ have named input variables $x = (x_1,\ldots,x_n)$ and output variable y. For $\alpha = \alpha(x)$ and $\beta = \beta(x,y)$, formulae of $L(\Sigma)$ having x and y as their free variables, we call $\{\alpha\}S\{\beta\}$ an *i/o asserted program* - i/o reads input-output, of course - the semantics of which is defined by

$A \models \{\alpha\}S\{\beta\}$ if, and only if, for all $a \in A^n$,

$$A \models \alpha(a) \rightarrow [(S(a)\downarrow \;\&\; \beta(a,S(a)\downarrow)) \vee S(a)\uparrow]$$

where A is a Σ-structure.

For $S \in PROG(\Sigma)$ and K a class of Σ-structures, the new *first-order i/o partial correctness theory of* S *over* K is defined to be the set of preconditions and postconditions for i/o asserted programs

$$I/O\text{-}PC_K(S) = \{(\alpha,\beta): \alpha,\beta \in L(\Sigma) \text{ and for each } A \in K,\ A \models \{\alpha\}S\{\beta\}\}.$$

Let us postpone any comments on this modification until we have seen what it achieves. Lemma 2.2 now disappears from the discussion:

5.1 TERMINATION LEMMA. *Let* K *be any class of* Σ*-structures. If* $P,Q \in PROG(\Sigma)$ *define total functions on each* $A \in K$ *then*

$$I/O\text{-}PC_K(P) = I/O\text{-}PC_K(Q) \textit{ implies } P \equiv_K Q.$$

PROOF. Suppose $P \equiv_K Q$ and let $A \in K$, $a \in A^n$ be such that $P(a) \neq Q(a)$ in A. Assume P(a) terminates in t steps. By the Definability Lemma 1.3, we can

encode the computation P(a) into the formula $COMP_{P,t}(x)$ and polynomial $OUT_{P,t}(x)$ over $L(\Sigma)$ so that for any $b \in A^n$, $A \models COMP_{P,t}(b)$ if, and only if, $|P(b)| \leq t$ and $P(b) = OUT_{P,t}(b)$. Define

$$\Theta(x,y) \equiv COMP_{P,t}(x) \rightarrow y = OUT_{P,t}(x).$$

It is easy to check that (true, $\Theta(x,y)$) lies in $I/O\text{-}PC_K(P)$ but not in $I/O\text{-}PC_K(Q)$. Q.E.D.

Thus, the i/o correctness theories determine the semantics of the everywhere terminating programs of *any* programming system (without recourse to hidden functions). Indeed, when the two kinds of correctness theory are compared, one finds that it is the issues to do with convergence which distinguish them.

Let $A = \{(\alpha,\beta): \alpha = \alpha(x), \beta = \beta(y) \in L(\Sigma)\}$ the set of all preconditions and postconditions for assertions. Then

$$PC_K(S) = I/O\text{-}PC_K(S) \cap A$$

and so

5.2 LEMMA. *Let* K *be any class of* Σ*-structures. For any* $P,Q \in PROG(\Sigma)$, *if* $I/O\text{-}PC_K(P) = I/O\text{-}PC_K(Q)$ *then* $PC_K(P) = PC_K(Q)$.

That the converse of Lemma 5.2 is false follows from Lemma 2.2 and 5.1, of course. The following basic connections between the correctness theories - all to do with termination properties - we prove in an appendix. Recall the Convergency Lemma 2.4; this now becomes

5.3 LOCALISATION LEMMA. *Let* K *be any class of* Σ*-structures. The following statements are equivalent:*

(i) *for all* $P,Q \in PROG(\Sigma)$,

$$I/O\text{-}PC_K(P) = I/O\text{-}PC_K(Q) \text{ implies } P \equiv_K Q;$$

(ii) *for any finite extension of* Σ *by constants,* $\Sigma(\underline{c}) = \Sigma \cup \{\underline{c}_1,\ldots,\underline{c}_n\}$,

and for any closed program S *over* $\Sigma(\underline{c})$, *if* S *diverges on some algebra in* K *then there is a sentence* Θ, *first-order over* $\Sigma(\underline{c})$, *which is satisfied in some algebra in* K *and such that for any* $A \in K$ *and* $a_1,\ldots,a_n \in A$, $(A,a_1,\ldots,a_n) \models \Theta \to S\uparrow$.

In stating the next results we bring in the *i/o total correctness theories:* let K be any class of Σ-structures and let $S \in PROG(\Sigma)$. Define

$$I/O\text{-}TC_K(S) = \{(\alpha,\beta)\text{: for all } A \in K,\ a \in A^n,\ A \models \alpha(a) \to [S(a)\downarrow \wedge \beta(a,S(a))]\}$$

5.4 LEMMA. *Let* K *be any class of* Σ*-structures. For any* $P,Q \in PROG(\Sigma)$, *if* $I/O\text{-}TC_K(P) = I/O\text{-}TC_K(Q)$ *then*

$$PC_K(P) = PC_K(Q) \text{ if, and only if, } I/O\text{-}PC_K(P) = I/O\text{-}PC_K(Q).$$

5.5 LEMMA. *There exists a class* K *and programs* P *and* Q *for which*

$$I/O\text{-}PC_K(P) = I/O\text{-}PC_K(Q) \text{ but } I/O\text{-}TC_K(P) \neq I/O\text{-}TC_K(Q).$$

The decision to use the $PC_K(S)$'s at the start of our investigation was made so as to conform with the standard practice of the literature on partial correctness (APT [1]); and these are the correctness theories used by MEYER and HALPERN [28]. The change to the i/o correctness theories is dictated by Lemma 2.2, but it hardly represents a less natural means of formulating Floyd's principle to require assertions to remember inputs when speaking of outputs. Indeed, Lemmas 5.1 and 5.3 suggest the opposite to be true. We should also say that using i/o correctness theories from the beginning would only have weakened our Basic Extension Theorem 3.1 and made not a guilder's worth of difference to the difficulty of proving Theorem 4.1.

PROGRAMMING SYSTEMS WITH SPECIFICATIONS

We are now going to consider programming systems $[K, PROG(\Sigma)]$ with K a class of Σ-structures syntactically defined by a set of axioms from a data

type specification language. This language we take to be $L(\Sigma)$ with its usual semantics in model theory, that based on Tarski's notion of satisfaction. Thus, our programming systems will be entirely syntactic objects of the form $[(\Sigma,E), PROG(\Sigma)]$, where E is a set of sentences of $L(\Sigma)$, and their input-output semantics will be based on $K = ALG(\Sigma,E)$, the class of all Σ-structures satisfying the axioms in E; such K are called *first-order axiomatisable classes*.

How does this description connect with those of the literature on the syntax and semantics of data types? All current work on data type specification uses first-order specifications (Σ,E) and their semantics $ALG(\Sigma,E)$. However, it is there common to want to define the meaning of (Σ,E) as a *particular structure in* $ALG(\Sigma,E)$, *unique up to isomorphism*. This arises quite naturally from the widely held informal view of data types as objects composed of different kinds of data domains on which are prescribed a number of primitive operations: the meaning of a data type τ becomes an algebraic structure $A(\tau)$ and a specification (Σ,E) of type τ is accepted as *correct* if its semantics is an algebra $A(\Sigma,E)$ isomorphic to $A(\tau)$. At best, logical semantics is able to define structures uniquely up to *elementary equivalence* only and this is far weaker than isomorphism. Thus, in working with data type specification problems *in isolation*, one refines the satisfaction semantics of (Σ,E) to (usually) its *initial algebra semantics* (ADJ [15]) and (sometimes) its *final algebra semantics* ([6, 35]) whose purpose it is to pick out a structure from $ALG(\Sigma,E)$ as the meaning of (Σ,E). The nature of these new semantic mechanisms need not concern us here though it is useful to point out that they impose conditions on the syntactical structure of the axioms E: the axioms are usually required to be equations or conditional equations. (A *partial* explanation of this is Corollary 3.2.5 in CHANG & KEISLER [7].) The crucial point, then, in assessing the relevance of our treatment of data types, lies not with the essentially algebraic problem of the correctness of data type specifications, but with the logical problem of proving partial correctness for programs relying on these specifications, independently of whether they are correct or not.

Mathematically, the proof theory of partial correctness for $[(\Sigma,E), PROG(\Sigma)]$ must be built up from first-order components: assertions from $L(\Sigma)$ with the axiom oracle for the Rule of Consequence taken as $Thm(\Sigma,E)$, the

set of all sentences of $L(\Sigma)$ provable from the axioms of E by the rules of first-order logic. Whatever the data type semantics intended for the programming system through the specification (Σ,E) might be, the proof theory for partial correctness is obliged to deal with the satisfaction semantics $ALG(\Sigma,E)$ as that of the programming system.

Our formal model of a programming system allows an equally perspicuous description of the standard treatment of Hoare's logic initiated by COOK [2] where the words data type and specification are not mentioned. There, one gives general rules of inference for the control structures of $PROG(\Sigma)$ and completes the construction of the proof system by fixing an interpretation A and taking Th(A) as the axiom oracle for the Rule of Consequence. This corresponds to our description of the canonical Hoare logic of the programming system $[(\Sigma, Th(A)), PROG(\Sigma)]$ where Th(A) acts as a data type specification. But the semantics of $(\Sigma,Th(A))$ is *not* the singleton class $\{A\}$; it is the class $ALG(\Sigma,Th(A))$ which contains many structures not isomorphic to A. The fundamental example of this is provided by the standard model of arithmetic $\underline{N}$, of course.

To return, for a moment, to the situation for algebraic specifications, the best that can be arranged is a partition of the specification into an algebraic part (Σ,E_A) which correctly defines the data type semantics A, uniquely up to isomorphism by, say, initial algebra semantics, together with a proof theoretical part (Σ,E'_A) chosen to make up the proofs of the correctness of programs of interest. This E'_A must be a subset of Th(A), and for the intended system $[A, PROG(\Sigma)]$ the strongest proof theory possible will be that of $[ALG(\Sigma,Th(A)), PROG(\Sigma)]$.

So it is then, that the study of the determinateness problem must contend with complex model-theoretic classes as representing the semantics of data types even when addressing computations on essentially simple minded data types such as arithmetic.

We will now consider determinateness for programming systems with (1) arbitrary first-order data type specifications; (2) algebraically styled specifications; and (3) complete first-order specifications which cover the $(\Sigma,Th(A))$ specifications described above.

GENERAL FIRST-ORDER SPECIFICATIONS

The i/o correctness theories fail to determine program equivalence not only for the first-order specified programming systems in general but for those with algebraic specifications and for those with complete first-order specifications. Thus, here we begin by proving a useful structural fact about the determinateness problem.

5.6 COUNTABILITY LEMMA. *Let* K *be a first-order axiomatisable class of* Σ*-structures and let* K_0 *be the subclass of* K *composed of all its countable structures. Then for any* $P,Q \in PROG(\Sigma)$

$$P \equiv_{K_0} Q \text{ if, and only if, } P \equiv_K Q.$$

Moreover, for any $S \in PROG(\Sigma)$,

$$I/O\text{-}PC_{K_0}(S) = I/O\text{-}PC_K(S).$$

To obtain this we look again at the local structure of computations.

Let K be *any* class of Σ-structures. A Σ-structure A is said to be *locally a* K*-structure* if each finite subset of A is contained within a Σ-substructure of A which belongs to K; write L(K) for the class of all locally K-structures.

5.7 LEMMA. *Let* K *be a class of* Σ*-structures. Then for* $P,Q \in PROG(\Sigma)$

$$P \equiv_{L(K)} Q \text{ if, and only if, } P \equiv_K Q.$$

PROOF. Now $P \equiv_{L(K)} Q$ implies $P \equiv_K Q$ because $K \subset L(K)$. Conversely, assume $P \equiv_K Q$. Let $A \in K$ and consider an arbitrary computation of P,Q on $a = (a_1,\ldots,a_n) \in A^n$. If B is a substructure of A containing $\{a_1,\ldots,a_n\}$ then, by the Locality of Computation Lemma 1.1,

$$P(a) \equiv_A Q(a) \text{ if, and only if, } P(a) \equiv_B Q(a).$$

Thus, $P \equiv_K Q$ implies $P \equiv_{L(K)} Q$. Q.E.D.

PROOF OF LEMMA 5.6. Obviously, $P \equiv_K Q$ implies $P \equiv_{K_0} Q$ as $K_0 \subset K$. By Lemma 5.7, $P \equiv_{K_0} Q$ implies $P \equiv_{L(K_0)} Q$: we show that $K \subset L(K_0)$. Let $A \in K$ and $a_1,\ldots,a_n \in A$. From a Downward Löwenheim-Skolem argument (for example, Theorem 3.1.6 in CHANG & KEISLER [7]), we may deduce there is a countable elementary substructure A_0 of A containing $a_1,\ldots,a_n$ which is a K-structure as K is axiomatisable. Since $A_0 \in K_0$ we have that $A \in L(K_0)$.

With regard to the last statement of the lemma, note that $I/O\text{-}PC_K(S) \subset I/O\text{-}PC_{K_0}(S)$ because $K_0 \subset K$. Assume for a contradiction that (α,β) lies in $I/O\text{-}PC_{K_0}(S)$ but not in $I/O\text{-}PC_K(S)$. Consequently, there is $A \in K$ and $a \in A^n$ such that $A \models \alpha(a)$ and $P(a)\downarrow$ but $A \not\models \beta(a,P(a))$. Since $P(a)\downarrow$ we can first-order express this computation:

$$A \models COMP_{S,t}(x) \rightarrow \neg\beta(x,OUT_{S,t}(x)).$$

Again by the Löwenheim-Skolem Theorem, there is a countable elementary substructure A_0 of A where

$$A_0 \models COMP_{S,t}(x) \rightarrow \neg\beta(x,OUT_{S,t}(x))$$

From this it follows, from propositional manipulations and the locality of computations, that $(\alpha,\beta) \notin I/O\text{-}PC_{K_0}(S)$, the required contradiction. Q.E.D.

ALGEBRAIC SPECIFICATIONS

Algebraic specifications are the simplest of the first-order specifications. This operates in their favour as far as the theory and practice of specifying data types is concerned, but against the needs of the subsequent proof theory. We will show that the i/o correctness theories fail to determine program equivalence for a very simple programming system with an algebraic specification.

5.8 THEOREM. *Let Σ be a signature composed of two unary functions* f,g *and a constant. Let* K *be the class of all Σ-structures satisfying the equations*

$$fg(x) = gf(x) = x.$$

Then there exist flow-chart programs $P,Q \in \mathrm{PROG}(\Sigma)$ *such that*

$$\text{I/O-PC}_K(P) = \text{I/O-PC}_K(Q) \text{ but } P \not\equiv_K Q.$$

The simplest possible counter-example is ruled out by the Singular Case Lemma 4.6 and the next simplest candidate is represented by our Conjecture 4.2. Having two function symbols with no axioms as a counter-example is ruled out by the Usual Case Lemma 4.4, so the variety defined in Theorem 5.8 is probably the best for our purpose. Notice that under their initial algebra semantics the equations define the integer arithmetic

$$(\mathbb{Z};\ 0,\ x+1,\ x-1).$$

PROOF OF THEOREM 5.8. Let P compute the two argument projection function $P(x,y) = x$ throughout K. For Q we require that

$$Q(x,y) = \begin{cases} x & \text{if } \langle x\rangle \text{ or } \langle y\rangle \text{ is finite or } x \in \langle y\rangle \text{ or } y \in \langle x\rangle \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Given the defining equations for K, it is straightforward to design a flow-chart program to play the rôle of Q. Clearly, $P \not\equiv_K Q$.

Assume, for a contradiction, that the i/o correctness theories are distinct. Since $\text{I/O-PC}_K(P) \subset \text{I/O-PC}_K(Q)$, let $\alpha = \alpha(x,y)$ and $\beta = (x,y,z) \in L(\Sigma)$ such that

$$K \models \{\alpha\}Q\{\beta\} \quad \text{and} \quad K \not\models \{\alpha\}P\{\beta\}.$$

Applying the known properies of P and Q, these expressions simplify to

$$K \models \alpha(x,y) \rightarrow [(Q(x,y)\downarrow \wedge \beta(x,y)) \vee Q(x,y)\uparrow]$$
$$K \not\models \alpha(x,y) \rightarrow \beta(x,y).$$

Set $\gamma(x,y) = \alpha(x,y) \wedge \neg\beta(x,y)$ and observe that for each $A \in K$, $a,b \in A$

$$A \models \gamma(a,b) \text{ implies } Q(a,b)\uparrow$$

and, therefore,

$$A \models \gamma(a,b) \text{ implies } \langle a\rangle \text{ and } \langle b\rangle \text{ are infinite, and } a \notin \langle b\rangle,\ b \notin \langle a\rangle.$$

Our hypotheses allow us to choose such an $A \in K$ and elements $a,b \in A$ and it is to this step we find a contradiction by means of the Compactness Theorem, Theorem 1.3.22 in CHANG and KEISLER [7].

Let $L = L(\Sigma)$ and add to it a constant symbol $\underline{\underline{a}}$ to obtain $L(\underline{\underline{a}})$. Then $(A,a) \models \gamma(\underline{\underline{a}},b)$ and for each $b \in (A,a)$, $(A,a) \models \gamma(\underline{\underline{a}},b)$ implies $Q(a,b)\uparrow$.

Set $T = Th(A,a)$, the set of all sentences of $L(\underline{\underline{a}})$ true in (A,a). Next we add a new constant symbol $\underline{\underline{b}}$ to $L(\underline{\underline{a}})$ and define the following set of sentences from $L(\underline{\underline{a}},\underline{\underline{b}})$:

$$T' = \{\neg\gamma(\underline{\underline{a}},\underline{\underline{b}}),\ \text{"}\langle b\rangle \text{ is infinite"},\ \text{"}\underline{\underline{b}} \notin \langle \underline{\underline{a}}\rangle\text{"}\}$$

It is easy to express the statements in quotation marks given the special definition of K.

By a routine application of the Compactness Theorem, the set of sentences $T \cup T'$ can be shown to have a model $B \in K$. In such a B there are elements a,b,c such that

$$B \models \gamma(a,b) \quad \text{and } B \models \neg\gamma(a,c).$$

We now use the following fact which is easy to prove from the specifications of K: if $A \in K$ and $a,b,c \in A$ are such that $\langle b\rangle$ and $\langle c\rangle$ are infinite, and a,b,c do not appear in one another's subalgebras, then there is some $\phi \in Aut(A)$ for which $\phi(a) = a$ and $\phi(b) = c$. Therefore, $b,c \in B$ can be exchanged, by an automorphism fixing a, in the pair of valid formulae above. And this is the sought for contradiction. Q.E.D.

COMPLETE FIRST-ORDER SPECIFICATIONS

By a *complete axiomatisable class* we mean the class $K = ALG(\Sigma,E)$ of all models of a first-order theory (Σ,E) having the property that for every sentence ϕ of $L(\Sigma)$ either ϕ or $\neg\phi$ is provable from E. By an *ω-categorical*

axiomatisable class we mean an axiomatisable class $K = ALG(\Sigma,E)$ having the property that any two countably infinite models in K are isomorphic.

Complete classes arise in two ways. First, the implicit specifications $(\Sigma,Th(A))$ in Hoare's logic are complete first-order theories; this is obvious. Secondly, the familiar numerical data types ω, the reals $\mathbb{R}$ and complex numbers $\mathbb{C}$ have natural first-order axiomatisations (Σ,E) in the theories of Presburger Arithmetic, real closed fields and algebraically closed fields, all of which are complete.

Programming systems whose data type semantics form such classes are particularly well characterised by their complete first-order specifications from the points of view of both proof theory and the theory of their *countable* models (see Section 2.3 of CHANG & KEISLER [7]; notice how several kinds of models distinguished by their morphism properties prove to be unique up to isomorphism). We will prove

5.9 THEOREM. *Let K be a complete axiomatisable class of Σ-structures and let $P,Q \in PROG(\Sigma)$. Then the following properties are equivalent.*

(i) $I/O\text{-}PC_K(P) = I/O\text{-}PC_K(Q)$;
(ii) *for some countable* $A \in K$, $P \equiv_A Q$;
(iii) *for some countable* $A \in K$, $I/O\text{-}PC_A(P) = I/O\text{-}PC_A(Q)$.

Yet, the i/o-correctness theories fail to determine program equivalence for complete classes. In the next section we will prove the following important fact:

5.10 THEOREM. *Let K be the class of all Σ_{arith} structures elementary equivalent to the standard model of arithmetic $\underline{\mathbb{N}}$. Then there exist $P,Q \in PROG(\Sigma_{arith})$ such that*

$$I/O\text{-}PC_K(P) = I/O\text{-}PC_K(Q) \text{ but } P \not\equiv_K Q.$$

Determinateness for complete classes can be neatly expressed in terms of a *logic of effective definitions* LED developed by TIURYN in [33] where it is equivalent to the condition on a class being $\forall$-LED *complete*.

Assuming the truth of Theorem 5.9 it is easy to obtain this positive result about ω-categorical axiomatisable classes, however.

5.11 COROLLARY. *Let* K *be an* ω*-categorical axiomatisable class of* Σ*-structures having an infinite element but having no finite elements. Then for any* $P,Q \in PROG(\Sigma)$

$$I/O\text{-}PC_K(P) = I/O\text{-}PC_K(Q) \text{ implies } P \equiv_K Q.$$

PROOF. Assume the correctness theories coincide. By the Łoś-Vaught Test (Theorem 3.1.10 in CHANG & KEISLER [7]), K is complete. By Theorem 5.9 there is a countably infinite structure $A \in K$ such that $P \equiv_A Q$. Let K_0 be the class of all countable K-structures. Since each structure in K_0 is isomorphic to A we know that $P \equiv_{K_0} Q$. Thus, by the Countability Lemma 5.6, $P \equiv_K Q$. Q.E.D.

PROOF OF THEOREM 5.9. First we prove that (1) implies (2). Now for $A \in K$, $P \equiv_A Q$ if, and only if, for no $a \in A^n$ any one of the following are true:

(i) for some $k,\ell \in \omega$, $A \models COMP_{P,k}(a) \wedge COMP_{Q,\ell}(a) \wedge OUT_{P,k}(a) \neq OUT_{Q,\ell}(a)$;

(ii) for some k, $A \models COMP_{P,k}(a)$ and for all $\ell \in \omega$, $A \not\models COMP_{Q,\ell}(a)$;

(iii) for some ℓ, $A \models COMP_{Q,\ell}(a)$ and for all $k \in \omega$, $A \not\models COMP_{P,k}(a)$

Clearly (i) is irrelevant for, in the presence of the hypothesis $I/O\text{-}PC_K(P) = I/O\text{-}PC_K(Q)$, when P and Q converge their outputs must coincide. Thus, we rephrase the situation as follows: let

$$T_{P,k}(x) = \{COMP_{P,k}(x);\ \neg COMP_{Q,\ell}(x) : \ell \in \omega\}$$

$$T_{Q,\ell}(x) = \{COMP_{Q,\ell}(x);\ \neg COMP_{P,k}(x) : k \in \omega\}$$

Then for any $A \in K$, $P \equiv_A Q$ if, and only if, no $a \in A^n$ *satisfies or realises either one of the types* $T_{P,k}(x)$, $T_{Q,\ell}(x)$. (This is standard terminology in model theory.)

To prove (2) we look for some countable $A \in K$ which *omits* these types.

Because K is complete we can apply the Extended Omitting Types Theorem (Theorem 2.2.15 in CHANG & KEISLER [7]) so that it is sufficient to prove K locally omits these types.

Suppose, for a contradiction, that $T_{P,k}(x)$ is locally realised. Then there is a formula θ consistent with K and such that

$$K \models \theta(x) \rightarrow COMP_{P,k}(x) \text{ and } K \models \theta(x) \rightarrow \neg COMP_{Q,\ell}(x) \text{ for } \ell \in \omega.$$

We claim the contradiction that $(\theta(x), \underline{\text{false}})$ lies in $I/O\text{-}PC_K(Q)$ but not in $I/O\text{-}PC_K(P)$. This is easy to see: let $A \in K$, $a \in A^n$. If $A \models \theta(a)$ then $Q(a)\uparrow$ and hence

$$A \models \theta(a) \rightarrow [Q(a)\downarrow \wedge \underline{\text{false}}] \vee Q(a)\uparrow.$$

Thus $K \models \{\theta\}Q\{\underline{\text{false}}\}$ and since, trivially, $A \models \theta(a)$ entails $P(a)\downarrow$ we have $K \not\models \{\theta\}P\{\underline{\text{false}}\}$. Applying the same argument to $T_{Q,\ell}(x)$ shows all the types are locally omitted and the implication is proved.

Now, that (2) implies (3) is obvious. And we conclude with a lemma which demonstrates that (3) implies (1).

5.12 LEMMA. *Let K be a complete axiomatisable class of Σ-structures and let $S \in PROG(\Sigma)$. Then for each $A \in K$*

$$I/O\text{-}PC_A(S) = I/O\text{-}PC_K(S).$$

PROOF. Since $A \in K$, $I/O\text{-}PC_K(S) \subset I/O\text{-}PC_A(S)$. For the reverse inclusion, suppose for a contradiction that the theories are distinct. There exist formulae $\alpha = \alpha(x)$, $\beta = \beta(x,y)$ such that $A \models \{\alpha\}S\{\beta\}$ but for some $B \in K$, $b \in B^n$ we have $S(b)\downarrow$ and $B \models \alpha(b) \wedge \neg\beta(b,S(b))$. Let $|S(b)| = t$ and define

$$\theta(x) \equiv \alpha(x) \wedge COMP_{S,t}(x) \wedge \neg\beta(x,OUT_{S,t}(x)).$$

Now, clearly, $B \models \theta(b)$ and $B \models \exists x.\theta(x)$. Since K is defined by a complete theory E we have $E \vdash \exists x.\theta(x)$. Therefore, as $A \models E$, we have that $A \models \exists x.\theta(x)$ which by the construction of θ contradicts $A \models \{\alpha\}S\{\beta\}$. Q.E.D.

6. ARITHMETIC PROGRAMS

The programs of $AP = PROG(\Sigma_{arith})$ we will henceforth call *arithmetic programs*. The purpose of arithmetic programs is to compute recursive functions on the set ω and, semantically, it seems reasonable to insist that one's interest in them is confined to the (unspecified) programming system

$[\underline{\underline{N}},AP]$ where $\underline{\underline{N}}$ is the standard model of arithmetic. This is not acceptable, however. Although we know, from Lemma 2.1, that the correctness theories determine program equivalence for $[\underline{\underline{N}},AP]$ we also know that Hoare logics for partial correctness do not operate without some first-order specification of $\underline{\underline{N}}$ acting as an interface between data type and proof theory. Thus the extensive collections of proof rules for programming constructs and the studies of their completeness properties in the monograph DE BAKKER [3], for example, pertain not to $[\underline{\underline{N}},AP]$ but to [CNT,AP] where CNT is the class of all models of $Th(\underline{\underline{N}})$, so called *complete number theory*. Here, of course, we have natural Hoare logics which are complete and so syntactically define the correctness theories but, in their turn, the correctness theories fail to determine program equivalence (Theorem 5.10 which we prove here). Even if this latter state of affairs were not the case then the fact that the specification $Th(\underline{\underline{N}})$ is not even arithmetical, having Turing degree 0^{ω}, forces a difficult compromise with our expectations about data type specifications.

Let us consider an alternate method of casting arithmetical computations in the form of a programming system. As is well-known, the class $\mathcal{WP}$ of all while-programs can compute all recursive functions on ω using a set of primitives smaller than Σ_{arith}; it is sufficient to use $\Sigma = \{0,SUCC\}$. The axioms of Presburger over Σ form a specification (Σ,E) which is recursive, complete and whose set of consequences is even decidable. Let $PrA = ALG(\Sigma,E)$. The first difficulty encountered by Floyd's principle is that $[PrA,\mathcal{WP}(\Sigma)]$ fails to possess any reasonable Hoare logic which is sound and complete for proving partial correctness (see BERGSTRA & TUCKER [5]).

Having introduced, and discounted, CNT and PrA as candidates fit to support arithmetical computation in a programming system, there is but one more first-order specification which ought to be tried: *Peano's axioms*. Let PA denote the class of all models of Peano arithmetic. Most regrettably, we have been unable to prove that the i/o-correctness theories determine program equivalence over PA. Thus, our first task is to offer an open problem and an opinion:

6.1 CONJECTURE. *For any arithmetic programs* P,Q

$$I/O\text{-}PC_{PA}(P) = I/O\text{-}PC_{PA}(Q) \text{ implies } P \equiv_{PA} Q.$$

We can provide, however, the following partial result.

Let $\pi_1(\underline{\underline{N}})$ be the set of all universal first-order sentences over Σ_{arith} true in the standard model of arithmetic $\underline{\underline{N}}$. Now let K be the subclass of those models in PA which, in addition, satisfy $\pi_1(\underline{\underline{N}})$. Clearly, CNT is a subclass of K.

6.2 THEOREM. *For any arithmetic programs* P,Q

$$\text{I/O-PC}_{PA}(P) = \text{I/O-PC}_{PA}(Q) \text{ implies } P \equiv_K Q.$$

Before proving Theorem 6.2, we shall explain why it is of any interest in these discussions. First, observe that it is easy to show

6.3 THEOREM. *For any arithmetical programs* P,Q

$$\text{I/O-PC}_{PA}(P) = \text{I/O-PC}_{PA}(Q) \text{ implies } P \equiv_{\underline{\underline{N}}} Q.$$

So the point at issue it that there is interesting information to be had about programs computing on $\underline{\underline{N}}$ by considering their behaviour over K. This next theorem will show that *programs on* $\underline{\underline{N}}$, *equivalent up to their input-output semantics over* $\underline{\underline{N}}$, *can be detected as operationally distinct over* $\underline{\underline{N}}$ *from the inequivalence of their input-output semantics over* K.

6.4 THEOREM. *Let* P,Q *be arithmetic programs. Suppose that* $P \equiv_{\underline{\underline{N}}} Q$ *but that* $P \not\equiv_K Q$. *Then the relative run times of* P *and* Q *over* $\underline{\underline{N}}$ *are unbounded in the sense that for any* $t \in \omega$ *there exists an input* $a \in A^n$ *such that*

$$\frac{|P(a)|}{|Q(a)|} + \frac{|Q(a)|}{|P(a)|} > t.$$

PROOF. First suppose that for some $A \in K$, $a \in A^n$ it is the case that $P(a)\downarrow$, $Q(a)\downarrow$ but $P(a) \neq Q(a)$. Let $|P(a)| = k$ and $|Q(a)| = \ell$ so that

$$A \models \text{COMP}_{P,k}(a) \wedge \text{COMP}_{Q,\ell}(a) \wedge \text{OUT}_{P,k}(a) \neq \text{OUT}_{Q,\ell}(a).$$

Now since all of $\pi_1(\underline{\underline{N}})$ is satisfied in A,

$$\underline{\underline{N}} \models \exists x.[\mathrm{COMP}_{P,k}(x) \wedge \mathrm{COMP}_{Q,\ell}(x) \wedge \mathrm{OUT}_{P,k}(x) \neq \mathrm{OUT}_{Q,\ell}(x)]$$

whence it follows that P,Q differ somewhere on $\underline{\underline{N}}$. As this contradicts $P \equiv_{\underline{\underline{N}}} Q$ we may assume that for some $A \in K$, $a \in A^n$ it is the case that $P(a)\downarrow$ but $Q(a)\uparrow$ (say).

Define $\theta_{k,\ell}(x) \equiv \mathrm{COMP}_{P,k}(x) \wedge \Lambda_{i=1}^{\ell} \neg\mathrm{COMP}_{Q,i}(x)$. If, again, $|P(a)| = k$ then for each $\ell \in \omega$, $A \models \exists x.\theta_{k,\ell}(x)$. As this is an existential sentence and $A \models \pi_1(\underline{\underline{N}})$ we deduce that $\underline{\underline{N}} \models \exists x.\theta_{k,\ell}(x)$ for each $\ell \in \omega$. Given t choose any $k,\ell \in \omega$ such that $\ell > tk$ and choose $a \in \underline{\underline{N}}^n$ such that $\underline{\underline{N}} \models \theta_{k,\ell}(a)$. Then

$$\frac{|P(a)|}{|Q(a)|} + \frac{|Q(a)|}{|P(a)|} \geq \frac{|Q(a)|}{|P(a)|} \geq \frac{\ell}{k} > \frac{tk}{k} = t. \qquad \text{Q.E.D.}$$

PROOF OF THEOREM 6.2. Contrapositively, assume $P \not\equiv_K Q$. If $P \not\equiv_{\underline{\underline{N}}} Q$ then we are done by Theorem 6.3; so assume $P \equiv_{\underline{\underline{N}}} Q$. This is the hypothesis of Theorem 6.4 and, using its proof, we may further assume that somewhere in K, P converges whilst Q diverges. Moreover, we can choose $k \in \omega$ such that for all $\ell \in \omega$, $\underline{\underline{N}} \models \exists x.\theta_{k,\ell}(x)$ where $\theta_{k,\ell}(x)$ is the formula defined above.

Let $\phi(z)$ be a formula such that (i) $\exists z.\phi(z)$ is satisfied somewhere in PA; and (ii) for any $M \in PA$, $m \in M$ if $M \models \phi(m)$ then m is a non-standard element of M. Such a formula exists by Gödel's Incompleteness Theorem.

There are now two cases to the proof, one of which must hold since $\exists z.\phi(z)$ is consistent with PA. Let $\mathrm{COMP}_Q(y,x)$ be a first-order representation the sequence $\{\mathrm{COMP}_{Q,\ell}(x) : \ell \in \omega\}$.

CASE 1: $\exists z.[\phi(z) \wedge \exists x[\mathrm{COMP}_{P,k}(x) \wedge (\forall y < z).\ \neg\mathrm{COMP}_Q(y,x)]]$ is satisfied in PA.

Then we claim that with

$$\alpha(x) \equiv \exists z.[\phi(z) \wedge \mathrm{COMP}_{P,k}(x) \wedge (\forall y < z).\ \neg\mathrm{COMP}_Q(y,x)]$$
$$\beta(x) \equiv \underline{\text{false}}$$

we have $PA \models \{\alpha\}Q\{\beta\}$ but $PA \not\models \{\alpha\}P\{\beta\}$. To see the first asserted program is valid is to notice its precondition can be satisfied, in which case it implies the divergence of Q. To see the second asserted program is not valid is to notice its precondition implies the convergence of P.

CASE 2: $\forall z.[\phi(z) \rightarrow \forall x.[COMP_{P,k}(x) \rightarrow (\forall y < z).\ COMP_Q(y,x)]]$ is satisfied in PA.

Let H(x) stand for "the least y, if any exist, such that $COMP_Q(y,x)$". Assuming, $M \in K$, $m \in M$ and $M \models \phi(m)$ then for any $a \in M^n$,

$$M \models COMP_{P,k}(a) \rightarrow H(a) \leq m$$

and $\sup\{H(a): M \models COMP_{P,k}(a)\}$ exists. Let this supremum be defined by formula $\gamma_t(s)$. Define $\alpha(x) \equiv \exists z.\phi(z) \wedge \exists s.[\gamma_k(s) \wedge (\forall y < s).\ \neg COMP_Q(y,x)]$.

We claim that PA $\models \{\alpha\}Q\{\underline{\text{false}}\}$ but PA $\not\models \{\alpha\}P\{\underline{\text{false}}\}$. Consider the first asserted program. The formula $\gamma_k(s)$ entails that s exceeds the lengths of all computations of Q on inputs satisfying $COMP_{P,k}(x)$. In particular, s exceeds all standard numbers as these computations may have arbitrarily large standard lengths on standard inputs. It follows that the precondition implies Q diverges and we are done.

On the other hand, the second asserted program is invalid in PA because $\exists x.\alpha(x)$ is satisfied and the precondition implies the convergence of P. Q.E.D.

To illustrate, in another way, the dependence of our problem on the specifications and the semantics they determine, we shall fix two arithmetic programs and consider their correctness theories through 4 changes of programming system. Let

$$P(x) \equiv x := 0$$
$$Q(x) \equiv \underline{\text{while}}\ x \neq 0\ \underline{\text{do}}\ x := PRED(x)\ \underline{\text{od}}$$

where PRED is the name reserved for the predecessor function on $\underline{N}$. Clearly, $P \equiv_{\underline{N}} Q$. Let $\Sigma_1 = \{0,PRED\} \subset \Sigma_{arith}$. For any class K of Σ_1-structures we have

$$I/O\text{-}PC_K(P) \subset I/O\text{-}PC_K(Q).$$

On the other hand, it is easy to think of K where $P \not\equiv_K Q$ because Q need not terminate. In the 4 systems to follow this will be so, but the correctness theories will not always remain distinct. This exercise with the vacillations of determinateness is a miniature of our study of specifications.

6.5 EXAMPLE: TRIVIAL SPECIFICATIONS

Consider P,Q as belonging to the system $[(\Sigma_1,\emptyset), \mathrm{PROG}(\Sigma_1)]$ and set $K_1 = \mathrm{ALG}(\Sigma_1)$ the class of all Σ_1-structures. When we studied "trivial" programming systems in Section 4 we were unable to settle determinateness in this case (remember Conjecture 4.2). However, the correctness theories of P,Q are distinct, if for no interesting reason: set $\alpha(x) \equiv \{x\neq 0 \wedge \mathrm{PRED}(x) = x\}$. Since for $A \in K_1$, $a \in A$, $A \models \alpha(a)$ forces $Q(a)$ to diverge we have

$$K_1 \models \{\alpha\}Q\{\underline{\mathrm{false}}\} \text{ but } K_1 \not\models \{\alpha\}P\{\underline{\mathrm{false}}\}.$$

6.6 EXAMPLE: ALGEBRAIC SPECIFICATIONS

Set $\Sigma_2 = \Sigma_1 \cup \{\mathrm{SUCC}\}$. We will turn our integer specification of Theorem 5.8 into a Horn formula specification of the natural numbers with successor and predecessor. Let E be the set of axioms

$$\mathrm{PRED}(0) = 0$$
$$X \neq 0 \rightarrow \mathrm{PRED}(X) \neq X$$
$$\mathrm{SUCC}(X) \neq 0$$
$$\mathrm{SUCC}(X) \neq X$$
$$\mathrm{PRED}(\mathrm{SUCC}(X)) = X$$
$$X \neq 0 \rightarrow \mathrm{SUCC}(\mathrm{PRED}(X)) = X.$$

The proof of Theorem 5.8 may be adapted, in a simple way, to prove the correctness theories of P and Q coincide.

6.7 EXAMPLE: PEANO ARITHMETIC

Consider P,Q as belonging to the system [PA,AP]. Although determinateness for this system is open it is easy to prove the correctness theories of P,Q are distinct. First note that P,Q differ on precisely the non-standard models in PA. By Gödel's Incompleteness Theorem, there is a formula $\alpha(x)$, consistent with PA, such that for $M \in \mathrm{PA}$, $m \in M$, $M \models \alpha(m)$ implies m is non-standard. It follows that

$$\mathrm{PA} \models \{\alpha\}Q\{\underline{\mathrm{false}}\} \text{ but } \mathrm{PA} \not\models \{\alpha\}P\{\underline{\mathrm{false}}\}$$

6.8 EXAMPLE: COMPLETE NUMBER THEORY

Consider P,Q as belonging to the system [CNT, AP] where CNT is the subclass of PA composed of those arithmetics satisfying $Th(\underline{\underline{N}})$. We will prove that

$$I/O\text{-}PC_{CNT}(P) = I/O\text{-}PC_{CNT}(Q)$$

which also proves Theorem 5.10.

Suppose, for a contradiction, the correctness theories are distinct. It is, by now, easy to see how to choose a formula $\alpha(x)$, consistent with CNT, such that for $M \in CNT$, $m \in M$, $M \models \alpha(m)$ implies $Q(m)\uparrow$. As $Th(\underline{\underline{N}})$ is a complete theory, $Th(\underline{\underline{N}}) \vdash \exists x.\alpha(x)$ and $\underline{\underline{N}} \models \exists x.\alpha(x)$. Thus, there is $n \in \underline{\underline{N}}$ such that $Q(n)\uparrow$ which by the definition of Q is impossible.

REFERENCES

[1] APT, K.R., *Ten years of Hoare's logic, a survey,* F.V. Jensen, B.H. Mayoh & K.K. Møller (eds.), *Proceedings from 5th Scandinavian Logic Symposium,* Aalborg University Press, Aalborg, 1979, 1-44.

[2] DE BAKKER, J.W., *Recursive procedures,* Mathematical Centre Tracts 24, Mathematical Centre, Amsterdam, 1973.

[3] ———————— , *Mathematical theory of program correctness,* Prentice-Hall International, London, 1980.

[4] BANACHOWSKI, L., A. KRECZMAR, G. MIRKOWSKA, H. RASIOWA, A. SALWICKI, *An introduction to algorithmic logic,* Banach Centre Publications 2 (1977) 7-99.

[5] BERGSTRA, J.A. & J.V. TUCKER, *The field of algebraic numbers fails to possess even a nice sound, if relatively incomplete, Hoare-like logic for its* while *- programs,* Mathematical Centre, Department of Computer Science Research Report IW 136, Amsterdam, 1980.

[6] BROY, M., W. DOSCH, H. PARTSCH, P. PEPPER & M. WIRSING, *Existential quantifiers in abstract data types,* in H.A. Maurer (ed.) *Proceedings Sixth International Colloquium on Automata, Languages and Programming,* Springer-Verlag, Heidelberg, 1979, 73-87.

[7] CHANG, C.C. & H.J. KEISLER, *Model theory,* North-Holland, Amsterdam, 1973.

[8] COOK, S.A., *Axiomatic and interpretative semantics for an ALGOL fragment,* Technical Report 79, Department of Computer Science, University of Toronto, 1975.

[9] ———— , *Soundness and completeness of an axiom system for program verification,* SIAM J. Computing 7 (1978) 70-90.

[10] DIJKSTRA, E.W., *A discipline of programming,* Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[11] ENGELER, E., *Algorithmic logic,* in J.W. de Bakker (ed.) *Foundations of computer science,* Mathematical Centre Tracts 63, Mathematical Centre, Amsterdam, 1975, 57-85.

[12] FENSTAD, J.E., *General recursion theory: an axiomatic approach,* Springer-Verlag, Berlin, 1980.

[13] FLOYD, R.W., *Assigning meaning to programs,* in J.T. Schwartz (ed.), *Mathematical aspects of computer science,* American Mathematical Society, Providence, Rhode Island, 1967, 19-32.

[14] FRIEDMAN, H., *Algorithmic procedures, generalized Turing algorithms, and elementary recursion theory,* R.O. Gandy & C.M.E. Yates (eds.), *Logic colloquium, '69,* North-Holland, Amsterdam, 1971, 316-389.

[15] GOGUEN, J.A., J.W. THATCHER & E.G. WAGNER, *An initial algebra approach to the specification, correctness and implementation of abstract data types,* in R.T. Yeh (ed.) *Current trends in programming methodology IV, Data structuring,* Prenctice-Hall, Engelwood Cliffs, New Jersey, 1978, 80-149.

[16] GREIBACH, S.A., *Theory of program structures: schemes, semantics, verification,* Springer-Verlag, Berlin, 1975.

[17] HAREL, D., *First-order dynamic logic,* Springer-Verlag, Berlin, 1979.

[18] HENNESSY, M., *A proof system for the first-order relational calculus,* J. Computer and Systems Science 20 (1980) 96-110.

[19] HOARE, C.A.R., *An axiomatic basis for computer programming,* Communications ACM 12 (1967), 576-580.

[20] ———————— , *Procedures and parameters: an axiomatic approach,* E. Engeler (ed.), *Symposium on the semantics of algorithmic languages,* Springer-Verlag, Berlin, 1971, 102-116.

[21] HOARE, C.A.R. & P. LAUER, *Consistent and complementary formal theories of the semantics of programming languages,* Acta Informatica 3 (1974), 135-155.

[22] HOARE, C.A.R. & N. WIRTH, *An axiomatic definition of the programming language PASCAL,* Acta Informatica 2 (1973) , 335-355.

[23] MANNA, Z., *The correctness of programs,* Journal of Computer and System Sciences 3 (1969), 119-127.

[24] ———————— , *Mathematical theory of computation,* McGraw-Hill, New York, 1974.

[25] MEYER, A.R., Letter to J. Tiuryn, 6th April 1979.

[26] ———————— , Letter to J. Tiuryn, 16th May 1979.

[27] MEYER, A.R. & I. GREIF, *Specifying program language semantics: a tutorial and critique of a paper by Hoare and Lauer,* Proceedings Sixth ACM Syposium on Principles of Programming Languages, ACM, New York, 1979, 180-189.

[28] MEYER, A.R. & J.Y. HALPERN, *Axiomatic definitions of programming languages. A theoretical assessment,* Proceedings Seventh ACM Symposium on Principles of Programming Languages, ACM, New York, 1980, 203-212.

[29] MOLDESTAD, J. & J.V. TUCKER, *On the classification of computable functions in an abstract setting,* in preparation.

[30] MOSCHOVAKIS, Y.N., *Abstract first-order computability* I, Transactions American Mathematical Society 138 (1969) 427-464.

[31] PARIKH, R., *Some applications of topology to program semantics,* to appear in Mathematical Systems Theory.

[32] SCHWARTZ, R., *An axiomatic semantic definition of* ALGOL 68, UCLA-ENG-7838, University of California at Los Angeles, Los Angeles, 1978.

[33] TIURYN, J., *Logic of effective definitions,* RWTH Aachen Department of Computer Science Research Report 55, Aachen, 1979.

[34] TUCKER, J.V., *Computing in algebraic systems,* in F.R. Drake & S.S. Wainer (eds.) *Recursion theory, its generalisations and applications,* Cambridge University Press, Cambridge, 1980.

[35] WAND, M., *Final algebra semantics and data type extensions,* J. Computer and Systems Science, 19 (1979) 27-44.

APPENDIX

In this appendix we prove the Localisation Lemma 5.3 and Lemmas 5.4, 5.5.

PROOF OF THE LOCALISATION LEMMA 5.3

Remember that the Convergency Lemma 2.4 said that the local condition (ii) in the statement of Lemma 5.3 on a class K was sufficient for the partial correctness theories $PC_K(S)$ to determine program equivalence over K. By Lemma 5.2, the condition is sufficient for the i/o correctness theories.

Assume the i/o correctness theories determine program equivalence over the class K. Let S be a closed program over the signature $\Sigma(\underline{c}) = \Sigma \cup \{\underline{c}_1, \ldots, \underline{c}_n\}$ and suppose S diverges somewhere in K. We must make a trivial case distinction between $n \neq 0$ and $n = 0$.

Let $n \neq 0$ and let $S = S_0(\underline{c}_1, \ldots, \underline{c}_n)$ where $S_0(x_1, \ldots, x_n)$ is a program over Σ with uninitialised input variables $x_1, \ldots, x_n$. We define $P, Q \in PROG(\Sigma)$ as programs abbreviating

$$P(x_1, \ldots, x_n) = x_1$$
$$Q(x_1, \ldots, x_n) = \underline{\text{if}}\ S_0(x_1, \ldots, x_n)\downarrow\ \underline{\text{then}}\ x_1\ \underline{\text{else}}\ \text{DIVERGE}\ \underline{\text{fi}}$$

Clearly, $P \not\equiv_K Q$ since P is everywhere convergent whereas Q is not because S_0 diverges by hypothesis. By their definition, $I/O\text{-}PC_K(P) \subset I/O\text{-}PC_K(Q)$ thus the determinateness assumption (i) implies there is a pair of formulae α, β such that

$$K \models \{\alpha\}Q\{\beta\} \quad \text{but} \quad K \not\models \{\alpha\}P\{\beta\}$$

The first-order sentence Θ required in condition (ii) is

$$\Theta \equiv \alpha(\underline{c}_1, \ldots, \underline{c}_n) \wedge \neg\beta(\underline{c}_1, \ldots, \underline{c}_n, \underline{c}_1)$$

the consistency and divergence property of which are easy to check. Q.E.D.

PROOF OF LEMMA 5.4

Lemma 5.3 yields one implication immediately without recourse to the hypothesis that the i/o total correctness theories agree.

Assume P,Q to be programs over K and that $I/O\text{-}TC_K(P) = I/O\text{-}TC_K(Q)$. We shall deduce that

$$PC_K(P) = PC_K(Q) \text{ implies } I/O\text{-}PC_K(P) = I/O\text{-}PC_K(Q).$$

Contrapositively, suppose there is some $\alpha = \alpha(x)$, $\beta = \beta(x,y)$ such that

$$K \models \{\alpha\}P\{\beta\} \quad \text{and} \quad K \not\models \{\alpha\}Q\{\beta\}.$$

Choose some $A \in K$ and $a \in A^n$ scuh that $Q(a)\downarrow$ and

$$A \models \alpha(a) \wedge \neg\beta(a,Q(a)).$$

Using the Definability Lemma 1.3 we can express the computation Q(a) in the first-order formula $COMP_{Q,t}(x)$ and polynomial $OUT_{Q,t}(x)$ for $t = |Q(a)|$:

$$A \models COMP_{Q,t}(a) \text{ and } A \models COMP_{Q,t}(b) \text{ implies } Q(b) = OUT_{Q,t}(b).$$

Notice that the pair

$$(COMP_{Q,t}(x),\ y = OUT_{Q,t}(x)) \in I/O\text{-}TC_K(Q).$$

Consider now the pair

$$(COMP_{Q,t}(x) \wedge \neg\beta(x,OUT_{Q,t}(x)),\ \underline{\text{false}})$$

Obviously, this pair does not belong to $PC_K(Q)$ because it is invalid on our chosen A. However it does lie in $PC_K(P)$. To see this let $B \in K$ and $b \in B^n$ and assume

$$B \models COMP_{Q,t}(b) \wedge \neg\beta(b,OUT_{Q,t}(b)).$$

The hypothesis on i/o total correctness theories implies

$$(\mathrm{COMP}_{Q,t}(x),\ y = \mathrm{OUT}_{Q,t}(x)) \in \text{I/O-TC}_K(P)\,.$$

Thus $P(b) = \mathrm{OUT}_{Q,t}(b)$ from which we may deduce

$$B \models \mathrm{COMP}_{Q,t}(b) \wedge \neg\beta(b,\mathrm{OUT}_{Q,t}(b)) \rightarrow [(P(b)\downarrow \wedge \underline{\text{false}}) \vee P(b)\uparrow]$$

Since B and b were arbitrarily chosen we are done. Q.E.D.

PROOF OF LEMMA 5.5

We want a class K and program P,Q over K whose i/o partial correctness theories agree but whose i/o total correctness theories are distinct. Example 6.7 will do nicely. The programs there defined over Peano Arithmetic we showed to have the same i/o partial correctness theories. To see that their i/o total correctness theories are not the same observe that (<u>true</u>,<u>true</u>) lies in $\text{I/O-TC}_K(P)$ but not in $\text{I/O-TC}_K(Q)$. Q.E.D.