stichting
mathematisch
centrum

MC

AFDELING INFORMATICA
(DEPARTMENT OF COMPUTER SCIENCE)

IW 160/81 FEBRUARI

J.A. BERGSTRA & J.V. TUCKER

HOARE'S LOGIC AND PEANO'S ARITHMETIC

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: 03D45 03D80 68B15 03D35 03D75 68B10
ACM-Computing Review-category 4.34,5.24

Hoare's logic and Peano's arithmetic[*)]

by

J.A. Bergstra[**)] & J.V. Tucker

ABSTRACT

We develop the proof theory of Hoare's logic for the partial correctness of *while*-programs applied to arithmetic as it is defined by Peano's axioms. By representing the strongest postcondition calculus in Peano arithmetic PA, we are able to show that Hoare's logic over PA is equivalent to PA itself.

KEY WORDS & PHRASES: *Hoare's logic, partial correctness, while programs, data type specifications, refinements, strongest postcondition calculus, Peano arithmetic*

[*)] This paper 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

INTRODUCTION

Hoare's logic is a formal system for the manipulation of statements about the partial correctness of *while*-programs; it was first described in HOARE [13] and studied in COOK [10]. The logic is a two-tiered axiomatic system for in addition to the axioms and proof rules for asserted programs there is an independent formal specification for the data types on which the programs are applied. The purpose of the specification is to generate the assertions about the data types necessary to govern the Rule of Consequence. Hoare's logic for the set $\mathcal{WP}$ of all *while*-programs with first-order assertion language L and first-order specification T we denote HL(T).

In this paper we consider the verification of programs computing on arithmetic N without the privilege that N is a structure given outright, and with the restriction that it must be axiomatically defined. Thus, whatever facts about arithmetic one needs in a program correctness proof must be formally deduced from a specification and not "popped" from the oracle Th(N), the first-order theory on N. We wish to study verification in Hoare's logic on an entirely proof-theoretic basis, founding proofs on what can be *derived* about arithmetic from what can be *stated about* N *in a specification.*

Peano's arithmetic PA is an ideal axiomatisation for this purpose. Seen from the point of view of a data type specification, one arrives at PA by first axiomatising the primitive operations of arithmetic in an algebraic way - indeed, the initial algebra semantics of these axioms picks out N as their unique meaning. And, secondly, by augmenting the specification with the induction scheme. The latter refinement means one can use any assertion about N which stems from its primitive operations and can be proved by induction. Viewed from the proof theory of Hoare's logic, the choice of PA combines conceptual simplicity and technical strength: although Gödel's Incompleteness Theorem tells that some valid assertions about N will not be provable from PA the fact is that meaningful assertions are decidedly difficult to find (see PARIS & HARRINGTON [18]).

The question we ask is simple enough: *How much of the semantical machinery which underlies partial correctness can be faithfully represented in the proof theoretical machinery of* PA *and* HL(PA)? We prove the following theorem in which we say that a specification T' *refines* a specification T if T $\vdash$ p implies T' $\vdash$ p for any assertion $p \in L$.

THEOREM. *Given an assertion* $p \in L$ *and program* $S \in \mathcal{WP}$ *one can effectively calculate an assertion* $SP(p,S) \in L$ *such that*

1. $SP(p,S)$ *defines the strongest postcondition of* S *relative to* p *on the set of states over* N.
2. $HL(PA) \vdash \{p\}S\{SP(p,S)\}$.

And, for any refinement T *of Peano arithmetic, including* PA *itself,*

3. $HL(T) \vdash \{p\}S\{q\}$ *if and only if,* $T \vdash SP(p,S) \rightarrow q$.

Strictly speaking, statement (1) is *not* of proof-theoretical interest: statements (2) and (3) establish the significance of the formula. Peano arithmetic provides a useful proof theory for partial correctness and (3) says that PA and HL(PA) are, in a very strong sense, equivalent systems. The corresponding theorem about weakest preconditions may also be proved, and the formalised pre- and postcondition calculus introduced can be used to deduce other pleasant results about HL(PA). A simple example of interest to us is the following stability theorem about refinements of Peano arithmetic.

Let R be a family of refinements of a data type specification T. Define the *core* of R by

$$CORE(R) = \{p \in L\colon T' \vdash p \text{ for each } T' \in R\}.$$

Clearly, $T \subset CORE(R)$.

COROLLARY. *Let* $\{p\}S\{q\}$ *be an asserted while-program and let* R *be a family of refinements of* PA *such that* $HL(T) \vdash \{p\}S\{q\}$ *for each* $T \in R$. *Then* R *is stable with respect to* $\{p\}S\{q\}$ *in the sense that* $HL(CORE(R)) \vdash \{p\}S\{q\}$. *In particular, if* $R_{p,S,q}$ *is the family of all refinements of* PA *which are capable of proving* $\{p\}S\{q\}$ *then* $CORE(R_{q,S,q})$ *proves* $\{p\}S\{q\}$ *too.*

The stability of refinement families is one of a number of questions which arise in the theoretical study of verification systems [14] especially those supporting data abstractions [16,17], where one is interested in the ways program correctness proofs depend upon specifications. Finite families of refinements are always stable, but stability remains a local property in general [8]. We shall prove this corollary here, since it is almost an immediate consequence of the theorem. Another application of the theorem appears in [9] where we prove a new kind of general completeness theorem for Hoare's logic, one which holds for arbitrary specifications rather than those which are semantically complete in the sense of COOK [10].

Sections 1 and 2 cover specifications and Hoare's logic. In Sections 3 and 4 we prove the theorem and its corollary. An acquaintance with HOARE [13] and COOK [10] is presumed, and the survey paper APT [1] is recommended. Some experience with proving formal theorems in a first-order logical theory is essential for the reader who wants to properly understand the vital calculations inside Peano arithmetic (these are confined to Section 3). Another relevant reference for the article is ZUCKER [20] where a careful proof of the expressiveness of L for recursive procedures on N can be found. An obvious problem is to turn Zucker's theorems about definability into proof-theoretical facts which generalise the main theorem here.

This paper belongs to a series of articles about Hoare's logic and specifications: various incompleteness and completeness properties of the logic are re-examined in [5,7,9]; algebraic specifications are studied in [6]; families of refinements are the subject of [8]. All these articles derive from [4], written with J. Tiuryn, and contain results pertinent to arithmetic computations, but none are prerequisite to understanding the mathematical contents of this paper.

Finally, we thank W. Hodges for useful information on the literature on Peano arithmetic.

1. ASSERTIONS, SPECIFICATIONS AND PROGRAMS

SYNTAX. First we summarise the syntactic ingredients of Hoare's logic.

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

If T is a set of assertions of L then the set of all formal theorems of

T is denoted Thm(T): we write $T \vdash p$ for $p \in \mathrm{Thm}(T)$. Such a set T of formulae is usually called a theory, but in the present context we obviously prefer the more suggestive term *specification*. Here L serves as both an assertion/program specification language and a data type specification language.

A specification T' is a *refinement* of a specification T if $\mathrm{Thm}(T) \subset \mathrm{Thm}(T')$. And two specifications T, T' are (*logically*) *equivalent* if $\mathrm{Thm}(T) = \mathrm{Thm}(T')$. If T is a specification and $R = \{T_i : i \in I\}$ is a family of refinements of T then the *core* of R by

$$\mathrm{CORE}(R) = \bigcap_{i \in I} \mathrm{Thm}(T_i)$$

Using the syntax of L, the set $\mathcal{WP} = \mathcal{WP}(\Sigma)$ of all <u>while</u>-programs over Σ is defined in the customary way.

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

SEMANTICS. Although semantics has no genuine rôle to play in this paper, some description of the meanings of the various components must be included because of statement (1) in the theorem, and in order to appreciate the use of Peano arithmetic as a data type specification.

For any structure A of signature Σ, the semantics of the first-order language L over Σ as determined by A has its standard definition in model theory and this we assume to be understood. The validity of $p \in L$ over structure A we write $A \models p$. The class of all models of a specification T is denoted Mod(T); we write $\mathrm{Mod}(T) \models p$ to mean that for every $A \in \mathrm{Mod}(T)$, $A \models p$. Gödel's Completeness Theorem says this about specifications:

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

As far as the proof theory of a data type axiomatisation T is concerned, the semantics of the specification *is* Mod(T). Before looking at Peano arithmetic and the special problems at hand, consider the algebraic specification methods for data types where one invariably has a *particular* semantic model in mind for a specification. Following ADJ [11], it is usual to settle on the initial model I(T) of Mod(T) as the unique meaning for an *algebraic* axiomatisation T. The logic of T is oblivious of this (or any other) particular choice because it yields only those facts true in *all* models of T. Refinements are a natural accessory of algebraic specifications: one starts with a simple algebraic specification (Σ,T) to establish the correctness of the desired data type semantics A and then adds to T various assertions true in A as the need arises in program correctness proofs (say). But refinements are an essential accessory of algebraic specifications for although the algebraic methods can define virtually any data type one wants, the kinds of assertion provable from algebraic formulae are rather restricted (see [8] for a thorough discussion of this problem.)

So consider Peano arithmetic in the light of these remarks. The desired data type semantics is the standard model of arithmetic N. The domain of N is the set ω of natural numbers and its primitive operations are the *successor function* x+1, *addition* x+y and *multiplication* x,y; $0 \in \omega$ is a distinguished element. We shall use these notations for the functions *and* the function symbols of its signature. Peano arithmetic PA is built up as follows:

Operator axioms: (1) $0 \neq x+1$

(2) $x+1 = y+1 \rightarrow x=y$

(3) $x+0 = x$

(4) $x+(y+1) = (x+y)+1$

(5) $x.0 = 0$

(6) $x.(y+1) = x.y + x$

Induction scheme: for each assertion $p \in L$, containing free variable x, the following is an axiom $[p(0) \wedge \forall x.(p(x) \rightarrow p(x+1))] \rightarrow \forall x.p(x)$.

Thus, we may observe that equations (3)-(6) alone define N under initial algebra semantics and so we may consider (1) and (2) as additions, making a first refinement of the standard algebraic specification for arithmetic, designed to rule out finite models. The theoretical objective of adding the induction scheme is self-evident and was alluded to in our introduction: *one wants to generate all assertions which make statements about* N *which can be based on its simple arithmetical operators and which can be proved by the principle of induction.* For example, one can obtain facts about the ordering $x \leq y$ of natural numbers by using the formula $\exists z.x+z = y$.

For the semantics of *WP* as determined by a structure A, we leave the reader free to choose any sensible account of while-program computations which applies to an arbitrary structure: COOK [10]; the graph-theoretic semantics in GREIBACH [12]; the denotational semantics described in DE BAKKER [2]. To the asserted programs we assign *partial correctness semantics*: the asserted program $\{p\}S\{q\}$ is *valid on a structure* A (in symbols: $A \models \{p\}S\{q\}$) if for each initial state $a \in States(A)$, $A \models p(a)$ implies either $S(a)$ terminates and $A \models q(S(a))$ or $S(a)$ diverges. And the asserted program $\{p\}S\{q\}$ is *valid for a specification* T if it is valid on *every* model of T; in symbols, $T \models \{p\}S\{q\}$ or $Mod(T) \models \{p\}S\{q\}$.

The *partial correctness theory of a structure* A is the set

$$PC(A) = \{\{p\}S\{q\}: A \models \{p\}S\{q\}\};$$

and the *partial correctness theory of a specification* T is the set

$$PC(T) = \{\{p\}S\{q\}: Mod(T) \models \{p\}S\{q\}\}.$$

Clearly,

$$PC(T) = \bigcap_{A \in Mod(T)} PC(A).$$

Finally, we define strongest postconditions. Let $p \in L$ and $S \in \mathcal{WP}$, both having n variables. The *strongest postcondition* of S and p on a structure A is the set

$$sp_A(p,S) = \{b \in A^n: \exists a \in A^n.[S(a) \text{ terminates in final state } b \text{ and } A \models p(a)]\}$$

1.1 LEMMA. $A \models \{p\}S\{q\} \iff sp_A(p,S) \subset \{b \in A^n: A \models q(b)\}$.

2. HOARE'S LOGIC

Hoare's logic for $\mathcal{WP} = \mathcal{WP}(\Sigma)$ with assertion language $L = L(\Sigma)$ and specification $T \subset L$, has the following axioms and proof rules for manipulating asserted programs: let $S,S_1,S_2 \in \mathcal{WP}$; $p,q,p_1,q_1,r \in L$; $b \in L$, a quantifier-free formula.

1. Assignment axiom scheme: for $e \in T(\Sigma)$ and x a variable of L, the asserted program

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

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

2. Composition rule:

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

3. Conditional rule:

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

4. Iteration rule:

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

5. Consequence rule:

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

And, in connection with 5,

6. Specification axiom: Each member of Thm(T) is an axiom.

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

2.1. REFINEMENT LEMMA. *Let* T *and* T' *be specifications. If* T' *is a refinement of* T *then* $HL(T) \subset HL(T')$. *Thus, if* T *and* T' *are equivalent specifications* *then* HL(T) = HL(T').

We shall need one derived rule of Hoare's logic.

2.2. DISJUNCTION LEMMA. *Let* T *be a specification. Then the following is a derived rule of* HL(T)

$$\frac{\{p_1\}S\{q_1\},\ldots,\{p_n\}S\{q_n\}}{\{p_1 v\ldots vp_n\}S\{q_1 v\ldots vq_n\}}$$

The corollary to Theorem 1 in COOK [10] says this:

2.3 SOUNDNESS THEOREM. *For any specification* T, HL(T) $\subset$ PC(T).

3. PROOF OF THE THEOREM: THE STRONGEST POSTCONDITION AND PEANO ARITHEMTIC

This section is devoted to making the *formal first-order strongest postcondition* SP(p,S) for a given assertion p and program S, and to proving some of its fundamental properties as a formula in Peano arithmetic. These fundamental properties are the Implication Law 3.4, the Existential law 3.5, and the Conjuction Law 3.6, and they are of proof-theoretical interest in their own right because they shape a formal calculus for the strongest postcondition within PA. Here they are needed to prove two theorems about invariant assertions for the while-construct: using Invariant Laws 3.7 and 3.8, the proofs of statements (2) and (3) of our theorem can be given as quite direct calculations in Section 4. However this section requires quite some time to digest. The reader may care to obtain an overview of the results of the section and then go on to consider the way the strongest postcondition calculus is used in Hoare's logic (Section 4). What makes difficulties in a proof of this theorem - and in a proof of an generalisation to more complicated program languages - is the extremely sharp picture of the *logical structure* of the strongest postcondition formulae one must have, if one is to get anything proved about them in PA. (The well-structured and mechanical appearance of formal proofs in PA should always be considered a criterion for the success of a logical analysis which PA is asked to support.)

We shall divide the work of this section between 3 unnumbered subsections.

THE DEFINITION OF SP(p,S). The formal strongest postcondition will be inductively defined over the structure of the program, and it will be obvious that SP(p,S) can be effectively calculated from p and S. The fact that SP(p,S) does indeed define the strongest postcondition $sp_N(p,S)$ on the standard model of arithmetic N will be a straightforward exercise whose interest or tediousness depends on the reader's chosen semantics for $\mathcal{WP}$. Because of the design of SP(p,S), statement (1) of our theorem will be trivial to verify for any sensible operational semantics for $\mathcal{WP}$.

We construct the formula SP(p,S) in a simple extension L_c of the first-order language L of PA. This language L_c merely contains formal names for encoding formulae which will be used in connection with the while-construct, and so represents a notational convenience. However, it is a notational convenience which must be justified, for its introduction immediately places us outside Peano arithmetic. To step back, we must also axiomatise the new function symbols in an extension PA_c of PA and observe that the theory PA_c based upon L_c is a so-called *eliminable extension* of PA based upon L (Theorem 3.1). Here is the construction of L_c and PA_c.

First add to L a binary function symbol C, to stand for a coding or pairing operation, together with unary function symbols L, R to stand for its left and right unpairing operations as expressed by the axiom

(1) $C(L(x),R(x)) = x$.

Next, we add to L two binary function symbols REDUCE and PROJECT to stand for special decomposition operations satisfying the expressions

$$\mathrm{REDUCE}(n,y) = R^n(y) \text{ and } \mathrm{PROJECT}(n,y) = LR^n(y)$$

These decompositions are formally axiomatised by the first-order formulae

(2) $\mathrm{REDUCE}(0,y) = y$

(3) $\mathrm{REDUCE}(x+1,y) = R(\mathrm{REDUCE}(x,y))$

(4) $\mathrm{PROJECT}(x,y) = L(\mathrm{REDUCE}(x,y))$

Thirdly, we add a ternary symbol INSERT to stand for an operation which introduces new codes into old ones. It is formally axiomatised by

(5) $\mathrm{INSERT}(x,0,z) = C(x,R(z))$

(6) $\mathrm{INSERT}(x,y+1,z) = C(L(x),\mathrm{INSERT}(x,y,R(z))$

Finally, we root the coding operation inside PA with the axiom

(7) $2.C(x,y) = (x+y).(x+y+1)+2y$
which is taken from the well-known bijection code: $\omega^2 \to \omega$ defined by

$$\text{code}(x,y) = \tfrac{1}{2}(x+y).(x+y+1)+y$$

Thus L_c is L augmented by the 6 new operations C,L,R, RED, PROJ, INS and PA_c is PA with axioms (1) - (7) *and with the induction scheme of* PA *modified to include all formulae of* L_c.

3.1. THEOREM. *The theory* PA_c *based on* L_c *is an eliminable extension of* PA *based on* L *in the sense that there is an effectively calculable map* $E:L_c \to L$ *such that*

(i) for each assertion $p \in L$, $E(p) = p$;

and for each assertion $p \in L_c$

(ii) $PA_c \vdash p \leftrightarrow E(p)$

(iii) if $PA_c \vdash p$ then $PA \vdash E(p)$.

The proof of a theorem such as Theorem 3.1 is an involved affair, one which unrewardingly copies the blueprint of §74 of KLEENE [15] (see also SMORYNSKI [19]). We omit the argument. Theorem 3.1 authorises us to use L_c to define our formulae SP(p,S), and prove formal properties about them using PA_c, while displaying L and PA in the statements of our theorems. For example, here is a lemma about codings in Peano which we will need later on. First, we introduce some important notations.

For $k \in \omega$, set

$$ROW_k(i,z) = (PROJ(0,PROJ(i,z)),\ldots,PROJ(k-2,PROJ(i,z)),RED(k-1,PROJ(i,z)))$$

For X a list of variables $x_1,\ldots,x_k$ we write

$$<X>_k = C(x_1,C(x_2,\ldots C(x_{k-1},x_k)\ldots))$$

3.2 LEMMA. *It is the case that*

(i) $PA \vdash j \le i \wedge z' = INS(u,i+1,z) \to (X = ROW_k(j,z) \leftrightarrow X = ROW_k(j,z'))$

(ii) $PA \vdash z' = INS(<X>_k,i+1,z) \to X = ROW_k(i+1,z')$

(iii) $PA \vdash X = ROW_k(i+1,z') \to \exists z.(z' = INS(<X>_k,i+1,z))$

Thus the formal theorems (i)-(iii) are actually written in L_c and proved using PA_c, but the elimination theorem maps each statement to an official statement in L provable from PA. The proof of Lemma 3.2 is left as an exercise for the reader.

We can now define SP(p,S), using this coding machinery to represent in L a operational account of its rôle on N.

Assume assertion $p \in L$ and program $S \in \mathcal{WP}$ are given. Let X denote the list of k program variables of S and let Y denote the list of those free variables of p not already contained in X. *This notation for the lists of variables in assertion* p *and program* S *we use without further declaration throughout the paper.* The formula SP(p,S) will have X and Y as its list of free variables and is inductively defined as follows:

$$SP(p, x := e) \equiv \exists y.[x=e[y/x] \wedge p[y/x]] \text{ where y is not a free variable of p.}$$

$$SP(p,S_1;S_2) \equiv SP(SP(p,S_1),S_2)$$

$$SP(p, \underline{\text{if}}\ b\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\ \underline{\text{fi}}) \equiv SP(p \wedge b,S_1) \vee SP(p \wedge \neg b,S_2)$$

$$SP(p, \underline{\text{while}}\ b\ \underline{\text{do}}\ S_0\ \underline{\text{od}}) \equiv INV(p,b,S_0) \wedge \neg b$$

where $INV(p,b,S_0)$ is the formula built up as follows.

First, set

$$A_p(i,z,Y) \equiv p[ROW_k(0,z)/X] \wedge \forall t < i.\ SP(X = ROW_k(t,z) \wedge b,S_0)[ROW_k(t+1,z)/X]$$

and then define

$$B_p(i,z,X,Y) \equiv A_p(i,z,Y) \wedge X = ROW_k(i,z)$$

Next set

$$INV^*(p,b,S_0)(i,X,Y) \equiv \exists z.B(i,z,X,Y)$$

and so define

$$INV(p,b,S_0)(X,Y) \equiv \exists i.INV^*(p,b,S_0)(i,X,Y).$$

THE STRONGEST POSTCONDITION CALCULUS. We give three formal theorems about the strongest postcondition formula.

3.4 IMPLICATION LAW. *Let* p,q *be assertions and* S *a program. Let* Z *be a list of variables containing the list* X *of the* k *program variables of* S. *Then*

$$PA \vdash \forall Z(p\rightarrow q) \rightarrow \forall Z(SP(p,S) \rightarrow SP(q,S))$$

and consequently

$$PA \vdash \forall Z(p\leftrightarrow q) \rightarrow \forall Z(SP(p,S) \leftrightarrow SP(q,S)).$$

PROOF. The argument is by induction on the structure of S for which the basis is the assignment.

Assignment, S ::= x := e. Clearly

$$PA \vdash \forall Z(p\rightarrow q) \rightarrow \forall Z.\forall y(p[y/x] \rightarrow q[y/x])$$

because x occurs in X ⊂ Z. Therefore,

$$PA \vdash \forall Z(p\rightarrow q) \rightarrow (\exists y(p[y/x] \wedge x=e[y/x]) \rightarrow \exists y.(q[y/x] \wedge x = e[y/x]))$$

which is

$$PA \vdash \forall Z(p\rightarrow q) \rightarrow \forall Z(SP(p,x := e) \rightarrow \forall Z(SP(q,x := e),$$

of course.

The induction step divides into 3 cases.

Composition, S ::= $S_1;S_2$. By the induction hypothesis applied to S_1 we know that

$$PA \vdash \forall Z(p\rightarrow q) \rightarrow \forall Z(SP(p,S_1) \rightarrow SP(q,S_2))$$

because Z contains the variables of S_1. By the induction hypothesis applied to S_2 we know that

$$\mathrm{PA} \vdash \forall Z(\mathrm{SP}(p,S_1) \rightarrow \mathrm{SP}(q,S_2)) \rightarrow \forall Z(\mathrm{SP}(\mathrm{SP}(p,S_1);S_2) \rightarrow \mathrm{SP}(\mathrm{SP}(q,S_1),S_2)).$$

Thus, by the definition of SP(p,S) and SP(q,S),

$$\mathrm{PA} \vdash \forall Z(p \rightarrow q) \rightarrow \forall Z(\mathrm{SP}(p,S_1;S_2) \rightarrow \mathrm{SP}(q,S_1;S_2)).$$

Conditional, S ::= <u>if</u> b <u>then</u> S_1 <u>else</u> S_2 <u>fi</u>. Clearly,

$$\mathrm{PA} \vdash \forall Z(p \rightarrow q) \rightarrow \forall Z(p \wedge b \rightarrow q \wedge b)$$

$$\mathrm{PA} \vdash \forall Z(p \rightarrow q) \rightarrow \forall Z(p \wedge \neg b \rightarrow q \wedge \neg b)$$

Hence, by the induction hypothesis,

$$\mathrm{PA} \vdash \forall Z(p \wedge b \rightarrow q \wedge b) \rightarrow \forall Z(\mathrm{SP}(p \wedge b,S_1) \rightarrow \mathrm{SP}(q \wedge b,S_1)$$

$$\mathrm{PA} \vdash \forall Z(p \wedge \neg b \rightarrow q \wedge \neg b) \rightarrow \forall Z(\mathrm{SP}(p \wedge \neg b,S_2) \rightarrow \mathrm{SP}(q \wedge \neg b,S_2))$$

because Z contains the variables of S_1 and S_2. Whence it follows that

$$\begin{aligned}\mathrm{PA} \vdash \forall Z(p \rightarrow q) \rightarrow \forall Z(&\mathrm{SP}(p \wedge b,S_1) \vee \mathrm{SP}(p \wedge \neg b,S_2) \\ &\rightarrow \mathrm{SP}(q \wedge b,S_1) \vee \mathrm{SP}(q \wedge \neg b,S_2))\end{aligned}$$

which is the theorem we require, by the definition of SP(p,S) and SP(q,S).

Iteration, S ::= <u>while</u> b <u>do</u> S_0 <u>od</u>. Because $X \subset Z$, we know that

$$\mathrm{PA} \vdash \forall Z(p \rightarrow q) \rightarrow \forall Z(p[\mathrm{ROW}_k(i,z)/X] \rightarrow q[\mathrm{ROW}_k(i,z)/X])$$

Conjoining formulae to make up $A_p(i,z,Y)$ and $A_q(i,z,Y)$ we deduce

$$\mathrm{PA} \vdash \forall Z(p \rightarrow q) \rightarrow \forall Z(A_p(i,z,Y) \rightarrow A_q(i,z,Y)).$$

Conjoining $X = ROW_k(i,z)$ and then $\exists i, \exists z$ we proceed to

$$PA \vdash \forall Z(p \to q) \to \forall Z(B_p(i,z,X,Y) \to B_q(i,z,X,Y))$$

$$PA \vdash \forall Z(p \to q) \to \forall Z(INV(p,b,S_0) \to INV(q,b,S_0)).$$

Finally, conjoining $\neg b$ it follows that

$$PA \vdash \forall Z(p \to q) \to \forall Z(SP(p,S) \to SP(q,S))$$

Notice we did not use the induction hypothesis in this case. Q.E.D.

3.5 EXISTENTIAL LAW. *Let* p *be an assertion and* S *a program. Let* z *be a variable which is not one in the list* X *of the program variables of* S. *Then*

$$PA \vdash SP(\exists z.p,S) \leftrightarrow \exists z.SP(p,S).$$

PROOF. The argument is by induction on the structure of S for which the basis is the assignment.

Assignment, S ::= x := e. By definition,

$$PA \vdash SP(\exists z.p,x{:=}e) \leftrightarrow \exists u(\exists z.p[u/x] \wedge x{=}e[u/x])$$

$$PA \vdash \exists u(\exists z.p[u/x] \wedge x{=}e[u/x]) \leftrightarrow \exists z.\exists u(p[u/x] \wedge x{=}[u/x])$$

$$PA \vdash SP(\exists z.p,x{:=}e) \leftrightarrow \exists z.SP(p,x{:=}e)$$

The induction step divides into 3 cases.

Composition, $S ::= S_1;S_2$. By definition,

$$PA \vdash SP(\exists z.p,S_1;S_2) \leftrightarrow SP(SP(\exists.p,S_1),S_2).$$

By the induction hypothesis applied to S_1 and the last statement of the Implication Law 3.4,

$$PA \vdash SP(SP(\exists z.p,S_1),S_2) \leftrightarrow SP(\exists z.SP(p,S_1),S_2)$$

and analogously with the induction hypothesis applied to S_2,

$$PA \vdash SP(\exists z.SP(p,S_1),S_2) \leftrightarrow \exists z.SP(SP(p,S_1),S_2)$$

Combining these theorems we conclude from the definition of SP(p,S),

$$PA \vdash SP(\exists z.p,S_1;S_2) \leftrightarrow \exists z.SP(p,S_1;S_2)$$

Conditional, S ::= <u>if</u> b <u>then</u> S_1 <u>else</u> S_2 <u>fi</u>. By definition,

$$PA \vdash SP(\exists z.p,S) \leftrightarrow SP(\exists z.p,S_1) \vee SP(\exists z.p,S_2)$$

By the induction hypothesis,

$$PA \vdash SP(\exists z.p,S_1) \vee SP(\exists z.p,S_2) \leftrightarrow (\exists z.SP(p,S_1)) \vee (\exists z.SP(p,S_2))$$

Thus, pulling out the existential quantifier and using the definition of SP(p,S) we derive

$$PA \vdash SP(\exists z.p,S) \leftrightarrow \exists z.SP(p,S).$$

Iteration, S ::= <u>while</u> b <u>do</u> S_0 <u>od</u>. By definition,

$$PA \vdash SP(\exists z.p,S) \leftrightarrow \exists i.\exists z'(A_{\exists z.p}(i,z',Y) \wedge X = ROW_k(i,z')) \wedge \neg b$$

Inspecting the definition of $A_{\exists z.p}(i,z',Y)$ one sees that

$$PA \vdash \exists i.\exists z'.A_{\exists z.p}(i,z',Y)) \leftrightarrow \exists i.\exists z'.\exists z.A_p(i,z',Y)$$

Whence the result follows since existential quantifiers commute:

$$PA \vdash SP(\exists z.p,S) \leftrightarrow \exists z(\exists i.\exists z'(A_p(i,z',Y) \wedge X = ROW_k(i,z')) \wedge \neg b)$$
$$PA \vdash SP(\exists z.p,S) \leftrightarrow \exists z.SP(p,S).$$

Notice we did not use the induction hypothesis in this case. Q.E.D.

3.6 CONJUCTION LAW. *Let* p,q *be assertions and* S *a program. Let the free variables of* q *and the program variables of* S *be disjoint lists. Then*

$$PA \vdash SP(p \wedge q,S) \leftrightarrow q \wedge SP(p,S)$$

The proof of this fact closely resembles the Existential Law 3.5 and is omitted.

THE INVARIANT LAWS We conclude our work with Peano arithmetic with two important laws about the invariants used in the inductive definition of the strongest postcondition in the iteration case. These laws are basic lemmas for the arguments in the next section.

3.7 INVARIANT LAW. *Let* p *be an assertion and let* S *be a program. Then*

(i) $PA \vdash p \rightarrow INV(p,b,S)$

(ii) $PA \vdash SP(INV(p,b,S) \wedge b,S) \rightarrow INV(p,b,S)$

3.8 INVARIANT LAW. *Let* p *be an assertion and let* S *be a program. Then*

(i) $PA \vdash INV^*(p,b,S)(0) \rightarrow p$

(ii) $PA \vdash INV^*(p,b,S)(i+1) \leftrightarrow SP(INV^*(p,b,S)(i) \wedge b,S)$

Now Invariant Law 3.8 is quite some work, but Invariant Law 3.7 is a short calculation once Law 3.8 is proven and so we give this proof first.

PROOF OF INVARIANT LAW 3.7. Consider case (i). Clearly,

$$PA \vdash p \rightarrow B_p(0,INS(<X>_k,0,z),X,Y)$$

$$PA \vdash B_p(0,INS(<X>_k,0,z),X,Y) \rightarrow INV^*(p,b,S)(0,X,Y)$$

$$PA \vdash INV^*(p,b,S)(0,X,Y) \rightarrow INV(p,b,S)$$

And we are done.

Consider case (ii).

$$PA \vdash SP(INV(p,b,S) \wedge b,S) \rightarrow SP(\exists i.INV^*(p,b,S)(i) \wedge b,S).$$

By the existential Law 3.5,

$$PA \vdash SP(\exists i.INV^*(p,b,S)(i) \wedge b,S) \rightarrow \exists i.SP(INV^*(p,b,S)(i) \wedge b,S).$$

By Invariant Law 3.8,

$$PA \vdash \exists i.SP(INV^*(p,b,S)(i) \wedge b,S) \rightarrow \exists i.INV^*(p,b,S)(i+1)$$

Trivially,

$$PA \vdash \exists i.INV^*(p,b,S)(i+1) \rightarrow INV(p,b,S)$$

And we are done. Q.E.D.

PROOF OF INVARIANT LAW 3.8. Case (i) is obvious so consider case (ii).

First of all we will need two formal theorems which we record here and prove at the end of the section.

3.9 LEMMA. *Let* p *be an assertion and* S *a program. Then*

(i) $PA \vdash A_p(i,z,Y) \leftrightarrow A(i,INS(u,i+1,z),Y)$

(ii) $PA \vdash A_p(i,z,Y) \wedge SP(X=ROW_k(i,z) \wedge b,S)[X/ROW_k(i+1,z)] \leftrightarrow A_p(i+1,z,Y)$

Here is the deduction for case (ii) of Invariant Law 3.8. By the definition of $INV^*(p,b,S)(i)$,

$$PA \vdash SP(INV^*(p,b,S)(i) \wedge b,S) \leftrightarrow SP(\exists z.B_p(i,z,X,Y) \wedge b,S)$$

By the Existential Law 3.5 and the definition of $B_p(i,z,X,Y)$, and the Conjunction Law 3.6,

$$PA \vdash SP(\exists z.B_p(i,z,X,Y) \wedge b,S) \leftrightarrow \exists z.[A_p(i,z,Y) \wedge SP(X=ROW_k(i,z) \wedge b,S)]$$

So consider this last formula through several transformations: it is equivalent in PA to

$$\exists z.\exists z'[z'=\mathrm{INS}(<X>_k,i+1,z)\wedge A_p(i,z,Y)\wedge SP(X=\mathrm{ROW}_k(i,z)\wedge b,S)]$$

By Lemma 3.9(i), it is equivalent to

$$\exists z.\exists z'[z'=\mathrm{INS}(<X>_k,i+1,z)\wedge A_p(i,z',Y)\wedge SP(X=\mathrm{ROW}_k(i,z)\wedge b,S)]$$

By Lemma 3.2(ii), it is equivalent to

$$\exists z.\exists z'[z'=\mathrm{INS}(<X>_k,i+1,z)\wedge A_p(i,z',Y)\wedge SP(X=\mathrm{ROW}_k(i,z)\wedge b,S) \wedge X=\mathrm{ROW}_k(i+1,z')]$$

Applying the Implication Law 3.4 to Lemma 3.2(i), it is equivalent to

$$\exists z.\exists z'[z'=\mathrm{INS}(<X>_k,i+1,z)\wedge A_p(i,z',Y)\wedge SP(X=\mathrm{ROW}_k(i,z')\wedge b,S) \wedge X=\mathrm{ROW}_k(i+1,z')]$$

And, clearly, this last formula is equivalent in PA to

$$\exists z.\exists z'[z'=\mathrm{INS}(<X>_k,i+1,z)\wedge A_p(i,z',Y)\wedge X=\mathrm{ROW}_k(i+1,z') \wedge SP(X=\mathrm{ROW}_k(i,z')\wedge b,S)[\mathrm{ROW}_k(i+1,z')/X]] \quad (*)$$

Now by Lemma 3.2(ii) and the definition of A_p, this formula *implies*

$$\exists z'.[A_p(i+1,z',Y)\wedge X=\mathrm{ROW}_k(i+1,z')] \quad (**)$$

which is equivalent to

$$\exists z'.B_p(i+1,z',X,Y)$$

which is equivalent to

$$\mathrm{INV}^*(p,b,S)(i+1).$$

On the other hand to prove the reverse implication, that (**) implies (*) in PA, one can rely on Lemma 3.2(iii).

This concludes the proof of Invariant Law 3.8, given Lemma 3.9.

<u>PROOF OF LEMMA 3.9</u>. Consider (i). By definition, $A_p(i,z,Y)$ is equivalent in PA to

$$p[ROW_k(0,z)/X] \wedge \forall t<i.SP(X=ROW_k(t,z)\wedge b,S)[ROW_k(t+1,z)/X]$$

Now by Lemma 3.2(i), the first conjunct can be replaced by

$$p[ROW_k(0,INS(u,i+1,z))/X].$$

By Implication Law 3.4, applied to Lemma 3.2(i), the second conjunct can be replaced by

$$\forall t < i.SP(X=ROW_k(t,INS(u,i+1,z))\wedge b,S)[ROW_k(t+1,z)/X]$$

Using Lemma 3.2(i) again, this formula is equivalent to

$$\forall t<i.SP(X=ROW_k(t,INS(u,i+1,z)\wedge b,S)[ROW_k(t+1,INS(u,i+1,z))/X]$$

and so the conjunction is what we require: by definition,

$$PA \vdash A_p(i,z,Y) \leftrightarrow A_p(i,INS(u,i+1,z),Y)$$

Next consider (ii). By definition, $A_p(i+1,z,Y)$ is equivalent in PA to

$$p[ROW_k(0,z)/X] \wedge \forall t<i+1.SP(X=ROW_k(t,z)\wedge b,S)[ROW_k(t+1,z)/X]$$

The second conjunct can be rewritten as

$$\forall t<i.SP(X=ROW_k(t,z)\wedge b,S)[ROW_k(t+1,z)/X] \wedge SP(X=ROW_k(i,z)\wedge b,S)$$
$$[ROW_k(i+1,z)/X]$$

And so regrouping the formula we immediately get

$$PA \vdash A_p(i+1,z,Y) \leftrightarrow A_p(i,z,Y) \wedge SP(X=ROW_k(i,z) \wedge b,S)[ROW_k(i+1,z)/X]$$

This concludes the proof of Lemma 3.9 and so the proof of the Invariant Laws 3.8 and 3.7.

4. PROOF OF THE THEOREM: THE STRONGEST POSTCONDITION AND HOARE'S LOGIC

It now remains for us to consider the rôle of a formal first-order strongest postcondition SP(p,S) in Hoare's logic HL(PA) based on Peano Arithmetic PA. The proofs of statements (2) and (3) of the theorem use induction on the structure of a program and are fairly smooth arguments because of the Invariant Laws which organise the calculations involving the while-construct.

STATEMENT 2. *For any* $p \in L$ *and* $S \in \mathcal{WP}$

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

PROOF. The argument is an induction on S for which the basis is the assignment statement.

Assignment: $S \equiv x := e$. First observe the following trivial theorems of Peano Arithmetic:

$$PA \vdash p \rightarrow (e=e[x/x] \wedge p[x/x])$$

$$PA \vdash (e=e[x/x] \wedge p[x/x]) \rightarrow \exists y.(x=e[y/x] \wedge p[y/x])$$

$$PA \vdash \exists y.(x=e[y/x] \wedge p[y/x]) \rightarrow \exists y.(x=e[y/x] \wedge p[y/x])[e/x]$$

By the definition of the formal strongest postcondition we conclude that

$$PA \vdash p \rightarrow SP(p,x := e)[e/x].$$

The axiom scheme for assignment provides

$$HL(PA) \vdash \{SP(p,x := e)[e/x]\}x := e\{SP(p,x := e)\}$$

$$\mathrm{HL(PA)} \vdash \{SP(p,x := e)[e/x]\}x := e\{SP(p,x := e)\}$$

and by the Rule of Consequence it follows that HL(PA) $\vdash$ $\{p\}S\{SP(p,S)\}$.

The induction step divides into 3 cases:

Composition: $S \equiv S_1;S_2$. The induction hypothesis applied to S_1 and S_2 yields that for any $p \in L$

$$\mathrm{HL(PA)} \vdash \{p\}S_1\{SP(p,S_1)\}$$

$$\mathrm{HL(PA)} \vdash \{SP(p,S_1)\}S_2\{SP(SP(p,S_1),S_2)\}$$

and the Composition Rule combines these formal theorems to derive

$$\mathrm{HL(PA)} \vdash \{p\}S_1;S_2\{SP(SP(p,S_1),S_2)\}$$

which is HL(PA) $\vdash$ $\{p\}S\{SP(p,S)\}$ by its definition.

Conditional: $S \equiv \underline{\text{if}}\ b\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\ \underline{\text{fi}}$. The induction hypothesis applied to to S_1 and S_2 yields that for any $p \in L$

$$\mathrm{HL(PA)} \vdash \{p \wedge b\}S_1\{SP(p\wedge b,S_1)\}$$

$$\mathrm{HL(PA)} \vdash \{p \wedge \neg b\}S_2\{SP(p \wedge \neg b,S_2)\}$$

From the derived rule Disjunction Lemma 2.2 and the Rule of Consequence it follows that

$$\mathrm{HL(PA)} \vdash \{p\wedge b\}S_1\{SP(p,b,S_1) \vee SP(p\wedge\neg b,S_2)\}$$

$$\mathrm{HL(PA)} \vdash \{p\wedge\neg b\}S_2\{SP(p\wedge b,S_1) \vee SP(p\wedge\neg b,S_2)\}$$

The Conditional Rule combines these formal theorems to derive

$$\mathrm{HL(PA)} \vdash \{p\}\ \underline{\text{if}}\ b\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\ \underline{\text{fi}}\ \{SP(p\wedge b,S_1) \vee SP(p\wedge\neg b,S_2)\}$$

which is HL(PA) $\vdash$ $\{p\}S\{SP(p,S)\}$ by its definition.

Iteration: $S \equiv \underline{\text{while}}\ b\ \underline{\text{do}}\ S_0\ \underline{\text{od}}$. The induction hypothesis applied to S_0 yields for any $p \in L$

$$HL(PA) \vdash \{INV(p,b,S_0) \wedge b\}S_0\{SP(INV(p,b,S_0) \wedge b,S_0)\}.$$

From Invariant Law 3.7(ii) and the Rule of Consequence it follows that

$$HL(PA) \vdash \{INV(p,b,S_0) \wedge b\}S_0\{INV(p,b,S_0)\}$$

and, using the Iteration Rule, that

$$HL(PA) \vdash \{INV(p,b,S_0)\}\ \underline{\text{while}}\ b\ \underline{\text{do}}\ S_0\ \underline{\text{od}}\ \{INV(p,b,S_0) \wedge \neg b\}.$$

Applying the Rule of Consequence with Invariant Law 3.7(i), and using the definition of the strongest post-condition, we conclude

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

This concludes the proof of the statement. Q.E.D.

STATEMENT 3. *For any* $p,q \in L$ *and* $S \in \mathcal{WP}$*, and for any extension* T *of Peano Arithmetic,*

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

PROOF. Assume $T \vdash SP(p,S) \rightarrow q$. Because T extends Peano Arithmetic, statement 2 implies $HL(T) \vdash \{p\}S\{SP(p,S)\}$; by the Rule of Consequence we derive $HL(T) \vdash \{p\}S\{q\}$.

The argument for the other implication is more involved and is an induction on S for which the basis is the assignment statement:

Assignment: $S \equiv x := e$. Suppose that $HL(T) \vdash \{p\}x := e\{q\}$. Then there must exist an assertion $r \in L$ such that

$$T \vdash p \rightarrow r[e/x]$$

$$HL(T) \vdash \{r[e/x]\}x := e\{r\}$$
$$T \vdash r \rightarrow q$$

Now in T we can calculate

$$T \vdash SP(p,x:=e) \rightarrow \exists y.(x=e[y/x] \wedge p[y/x]) \text{ by definition;}$$

$$T \vdash SP(p,x:=e) \rightarrow \exists y.(x=e[y/x] \wedge r[e[y/x]/x])$$

because from $T \vdash p \rightarrow r[e/x]$ it follows that $T \vdash p[y/x] \rightarrow r[e[y/x]/x]$. Continuing:

$$T \vdash SP(p,x:=e) \rightarrow \exists y.(x=e[y/x] \wedge r[x/x]$$

$$T \vdash SP(p,x:=e) \rightarrow \exists y.(x=e[y/x] \wedge r)$$

$$T \vdash SP(p,x:=e) \rightarrow r \qquad \text{because } y \notin FV(r);$$

$$T \vdash SP(p,x:=e) \rightarrow q$$

And this is what is required.

The induction step divides into 3 cases:

Composition: $S \equiv S_1;S_2$. Suppose that $HL(T) \vdash \{p\}S\{q\}$. Then there exists an assertion $r \in L$ such that

$$HL(T) \vdash \{p\}S_1\{r\} \text{ and } HL(T) \vdash \{r\}S_2\{q\}$$

Applying the induction hypothesis to S_1 we find that $T \vdash SP(p,S_1) \rightarrow r$ and by the Rule of Consequence it follows that $HL(T) \vdash \{SP(p,S_1)\}S_2\{q\}$. Now applying the induction hypothesis to this last asserted program yields

$$T \vdash SP(SP(p,S_1),S_2) \rightarrow q$$

which is $T \vdash SP(p,S) \rightarrow q$ by the definition of the strongest postcondition.

Conditional: $S \equiv \underline{\text{if}}\ b\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\ \underline{\text{fi}}$. Suppose that $HL(T) \vdash \{p\}S\{q\}$. Then

$$HL(T) \vdash \{p \wedge b\}S_1\{q\} \text{ and } HL(T) \vdash \{p \wedge \neg b\}S_2\{q\}.$$

Applying the induction hypothesis yields

$$T \vdash SP(p \wedge b, S_1) \rightarrow q \text{ and } T \vdash SP(p \wedge \neg b, S_2) \rightarrow q.$$

Thus,

$$T \vdash SP(p \wedge b, S_1) \vee SP(p \wedge \neg b, S_2) \rightarrow q$$

which is $T \vdash SP(p,S) \rightarrow q$ by the definition of the strongest postcondition.

Iteration: $S \equiv \underline{\text{while}}\ b\ \underline{\text{do}}\ S_0\ \underline{\text{od}}$. Suppose that $HL(T) \vdash \{p\}S\{q\}$. Then there must exist an assertion $r \in L$ such that

$$\begin{aligned} T &\vdash p \rightarrow r \\ HL(T) &\vdash \{r \wedge b\}S_0\{r\} \\ T &\vdash r \wedge \neg b \rightarrow q \end{aligned}$$

Applying the induction hypothesis to the asserted program above yields

$$T \vdash SP(r \wedge b, S_0) \rightarrow r \qquad (*)$$

We shall derive the following theorem in T

$$T \vdash INV(p,b,S_0) \rightarrow r \qquad (**)$$

whence we simply calculate

$$\begin{aligned} T &\vdash (INV(p,b,S_0) \wedge \neg b) \rightarrow (r \wedge \neg b) \\ T &\vdash SP(p,S) \rightarrow q \end{aligned}$$

by definition of the strongest postcondition and the fact that $T \vdash r \wedge \neg b \rightarrow q$.

To prove (**) first recall that

$$\mathrm{INV}(p,b,S_0) \equiv \exists i.\mathrm{INV}^*(p,b,S_0)(i)$$

and so it is sufficient to prove

$$T \vdash \mathrm{INV}^*(p,b,S_0)(i) \rightarrow r$$

This is done using the induction scheme in Peano Arithmetic which is also available in T.

Basis: $T \vdash \mathrm{INV}^*(p,b,S_0)(0) \rightarrow r$.

This follows from the Invariant Law 3.8(i) and $T \vdash p \rightarrow r$.

Induction Step: If $T \vdash \mathrm{INV}^*(p,b,S_0)(i) \rightarrow r$ then $T \vdash \mathrm{INV}^*(p,b,S_0)(i+1) \rightarrow r$

Consider Invariant Law 3.8(ii): the theorem of T we require follows from

$$T \vdash SP(\mathrm{INV}^*(p,b,S_0)(i) \wedge b,S_0) \rightarrow r$$

This follows easily from an application of Implication Law 3.4 to

$$T \vdash \mathrm{INV}^*(p,b,S_0)(i) \wedge b \rightarrow r \wedge b$$

$$T \vdash SP(\mathrm{INV}^*(p,b,S_0)(i) \wedge b,S_0) \rightarrow SP(r \wedge b,S_0) \quad \text{by}$$

$$T \vdash SP(\mathrm{INV}^*(p,b,S_0)(i) \wedge b,S_0) \rightarrow r \quad \text{by } (*)$$

This concludes the proof of (**), statement 3 and the theorem. Q.E.D.

PROOF OF COROLLARY. Let R be a family of refinements of PA such that for each $T \in R$ we have $HL(T) \vdash \{p\}S\{q\}$. Then, by the theorem, statement 3, we have $T \vdash SP(p,S) \rightarrow q$ for each $T \in R$ and so, by definition, the formula $SP(p,S) \rightarrow q \in CORE(R)$. Now PA is extended by CORE(R), thus $HL(CORE(R)) \vdash \{p\}S\{q\}$ by statement 3 of the theorem. Q.E.D.

REFERENCES

[1] APT, K.R., *Ten years of Hoare's logic, a survey* in F.V. JENSEN, B.H. MAYOH and K.K. MØLLER (eds), *Proceedings from 5th Scandinavian Logic Symposium,* Aalborg University Press, Aalborg, 1979, 1-44. (A second edition of this paper will appear in ACM Transactions on Programming Languages and Systems).

[2] DE BAKKER, J.W., *Mathematical theory of program correctness,* Prentice-Hall International, London, 1980.

[3] BARWISE, J., *Handbook of mathematical logic,* North-Holland, Amsterdam, 1977.

[4] BERGSTRA, J.A., J. TIURYN & J.V. TUCKER, *Floyd's principle, correctness theories and program equivalence,* Mathematical Centre, Department of Computer Science Research Report IW 145, Amsterdam, 1980. (To appear in Theoretical Computer Science.)

[5] BERGSTRA, J.A. & J.V. TUCKER, *Some natural structures which fail to possess a sound and decidable Hoare-like logic for their while-programs*. (To appear in Theoretical Computer Science. An earlier edition of this paper is registered at the Mathematical Centre as Report IW 136/80).

[6] BERGSTRA, J.A. & J.V. TUCKER, *Algebraically specified programming systems and Hoare's logic,* Mathematical Centre, Department of Computer Science Research Report IW 143, Amsterdam, 1980.

[7] BERGSTRA, J.A. & J.V. TUCKER, *Expressiveness and the completeness of Hoare's logic,* Mathematical Centre, Department of Computer Science Research Report IW 149, Amsterdam, 1980.

[8] BERGSTRA, J.A. & J.V. TUCKER, *On the refinement of specifications and Hoare's logic,* Mathematical Centre, Department of Computer Science Research Report IW 155, Amsterdam, 1980.

[9] BERGSTRA, J.A. & J.V. TUCKER, *On the completeness of Hoare's logic,* Mathematical Centre, Department of Computer Science Research Report Amsterdam, 1981. In preparation.

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

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

[11] 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*, Prentice-Hall, Englewood Cliffs, New Jersey, 1978, 80-149.

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

[13] HOARE, C.A.R., *An axiomatic basis for computer programming*, Communications Association Computing Machinery 12 (1969) 576-580.

[14] IGARASHI, S., R.L. LONDON & D.C. LUCKHAM, *Automatic program verification I: a logical basis and its implementation*, Acta Informatica 4 (1975) 145-182.

[15] KLEENE, S.C., *Introduction to metamathematics*, North-Holland/P. Noordhof, Amsterdam/Groningen 1952.

[16] LUCKHAM, D.C. & N. SUZUKI, *Verification of array, record and pointer operations in PASCAL*, ACM-Transactions on Programming Languages and Systems 1 (1979) 226-244.

[17] MUSSER, D.R., *Abstract data type specification in the AFFIRM system*, IEEE Transactions on software engineering 6(1) (1980) 24-32.

[18] PARIS, J. & L. HARRINGTON, *A mathematical incompleteness in Peano arithmetic*, in BARWISE [3] 1133-1142.

[19] SMORYNSKI, C., *The incompleteness theorems*, in BARWISE [3] 821-865.

[20] ZUCKER, J.I., *Expressibility of pre- and post- conditions*, in DE BAKKER [2] 444-465.