COMPLETENESS AND INCOMPLETENESS THEOREMS FOR HOARE-LIKE AXIOM SYSTEMS[+]

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment for the Degree of

Doctor of Philosophy

by

Edmund Melson Clarke, Jr.

September 1976

[+]This research was supported by an IBM Graduate Fellowship Award.

BIOGRAPHICAL SKETCH

Edmund Melson Clarke, Jr. was born on July 27, 1945 in Newport News, Virginia. In June 1967, he received a Bachelor of Arts Degree with High Distinction in Mathematics from the University of Virginia, and in August 1968, a Master of Arts Degree in Mathematics from Duke University. His professional experience includes serving on the faculty of the Department of Mathematics at Madison College from September 1968 until June 1972. In August 1974 he received a Master of Science Degree in Computer Science from Cornell University. His professional memberships include the Association of Computing Machinery and Phi Beta Kappa.

ACKNOWLEDGEMENTS

I am deeply indebted to Professor Robert Constable, my thesis advisor. His questions, critical suggestions, and interest made this work possible. His confidence in me and encouragement made it bearable.

I am grateful to Professors Juris Hartmanis and Richard Platek for serving on my special committee. Also I wish to thank Professors James Donahue and Susan Owicki and fellow graduate students, Mike O'Donnell and John Privitera, for helping me with various parts of this research.

Most of all I am grateful to my sons, Selden and Jon, whose mischievous ways brightened many a dark hour and to my wife Martha who "could make a chicken last three meals and still taste like a gourmet delight" and without whose love and encouragement this thesis would never have been completed.

TABLE OF CONTENTS

COMPLETENESS AND INCOMPLETENESS THEOREMS FOR
HOARE-LIKE AXIOM SYSTEMS

Edmund Melson Clarke, Jr., Ph.D.
Cornell University 1976

ABSTRACT

It is well known that Hoare-like deduction systems for establishing partial correctness of programs may fail to be complete because of (a) incompleteness of the assertion language relative to the underlying interpretation and (b) inability of the assertion language to express the invariants of loops. S. Cook has shown that if there is a complete proof system for the assertion language (e.g. all true statements of the assertion language) and if the assertion language satisfies a certain natural expressibility condition, then sound and complete axiom systems for a fairly large subset of Algol may be devised. We exhibit programming language constructs for which it is impossible to obtain sound and complete sets of Hoare-like axioms even in this special sense of Cook's. These constructs include (i) recursive procedures with procedure parameters in the presence of global variables and static scope and (ii) coroutines in a language which also allows recursive procedures. Such features appear to be inherently difficult to prove correct and (one might argue) should be avoided in the design of languages suitable for program verfication.

A fixed point characterization of partial and total correctness for recursive programs is also given. This characterization is based on a treatment of program invariants as fixed points of a predicate transformer which can be obtained in a natural manner from the program text. We show

that the weakest precondition for partial correctness is the maximal fixed point of this predicate transformer and that the weakest precondition for total correctness is the minimal fixed point. This characterization is important because it sheds light on the relationship between partial and total correctness and because it simplifies proofs of soundness and completeness. Non-regular recursions and total correctness are considered, and a theoretical justification is given for the claim that "total correctness is not substantially more difficult to establish than partial correctness."

CHAPTER 1

INTRODUCTION

1.1 Introduction

One of the most important problems facing users of digital computers is the problem of program reliability. An approach to this problem which has been extensively investigated in recent years is one in which programs are proved correct in much the same way that mathematical statements are proved correct; it should be possible to be as confident in correctness of a sorting program as in the truth of the Pythagorean theorem.

Many different formalisms have been proposed for proving programs correct. Of these probably the most widely referenced is the axiomatic approach of C. A. R. Hoare [H069]. Hoare gives a set of axioms and rules of inference for proving partial correctness of Algol-like programs. The formulas in Hoare's system are triples of the form $\{P\}\ A\ \{Q\}$ where A is a statement in the programming language and P and Q are predicates expressed in the language of the first order predicate calculus (the assertion language). The partial correctness assertion $\{P\}\ A\ \{Q\}$ is true iff whenever P holds for the initial values of the program variables and A is executed, then either A will fail to terminate or Q will be satisfied by the final values of the program variables.

Examples of axioms and rules of inference for programming constructs include:

(1) $\{P\ \frac{e}{x}\}\ x:=e\ \{P\}$ — assignment statement

(2) $\dfrac{\{P\}\ A_1\ \{S\},\ \{S\}\ A_2\ \{R\}}{\{P\}\ (A_1;A_2)\ \{R\}}$ — composition of statements

(3) $\dfrac{\{P \wedge b\}\ A\ \{P\}}{\{P\}\ \text{while } b \text{ do } A\ \{P \wedge \sim b\}}$ — while statement

The axioms are designed to capture the meanings of the basic statements of the programming language. Proofs of correctness for composite statements are constructed by using these axioms together with a proof system for the assertion language.

Modern programming languages use statements considerably more complicated than those described above. One might wonder how well Hoare's axiomatic approach can be extended to handle more complicated statements. In this paper we will be interested in the question of whether there are programming languages for which it is impossible to obtain good Hoare-like axioms. This question is of obvious importance in the design of programming languages whose programs can be naturally proved correct.

But what is a good Hoare-like axiom? One property a good axiom system should have is soundness ([HO74], [DO76]). A deduction system is sound iff every statement which is provable within the system is, indeed, true. Another property is completeness [CO75]. A deduction system is complete if every true statement is provable. One suspects from the Godel incompleteness theorem that, if the deduction system for the assertion language is axiomatizable and if a sufficiently rich interpretation (such as number theory) is used for the assertion language, then for any (sound) system of Hoare-like axioms, there will be assertions {P} A {Q} which are true but not provable within the system. One might wonder, however, if this incompleteness of the Hoare-like systems reflects some inherent complexity of the programming language constructs or whether it is due entirely to the incompleteness of the assertion language. If, for example, we are dealing with the integers, then for any consistent axiomatizable proof system there will be predicates which are true of the integers but not provable within the system. How can we talk about the completeness of a Hoare-like axiom system independently of its assertion language?

One way of answering this question is due to S. Cook [CO75]. He gives a Hoare-like axiom system for a subset of Algol including the while statement and non-recursive procedures. He then proves that if there is a complete proof system for the assertion language (e.g. all true statements of the assertion language) and if the assertion language satisfies a certain natural expressibility condition, then every true partial correctness assertion will be provable. Gorelick [GO75] extends Cook's work to handle recursive procedures. Other completeness results are given by deBakker and Meertens [DE73] and by Manna [MA70].

In this thesis we extend the work of Cook in two directions, first, toward general techniques for establishing the soundness and completeness of Hoare-like axiom systems; then toward the discovery of natural programming language constructs for which it is impossible to obtain good Hoare-like axiom systems.

In the first part of the thesis a fixed point characterization of partial and total correctness for recursive programs is given. This characterization is based on a treatment of program invariants as fixed points of a predicate transformer which can be obtained in a natural manner from the program text. We show that the weakest precondition for partial correctness is the maximal fixed point of this predicate transformer and that the weakest precondition for total correctness is the minimal fixed point. This characterization is important because it sheds light on the relationship between partial and total correctness and because it can be used to simplify proofs of soundness and completeness for programming language constructs. Non-regular recursions and total correctness are considered, and a theoretical justification is given for the claim that "total correctness is not substantially more difficult to establish than partial correctness."

Next we show that there are natural control mechanisms for which it is impossible to obtain sound and complete sets of axioms in the sense described above. While such incompleteness is expected for data structures (e.g. the integers, stacks, queues, etc.), it is surprising that is should exist for control structures.

The first programming language feature considered is recursive procedures with procedure parameters (provided that static scope is used and global variables are allowed). This result is surprising for two reasons. First, it holds even if we disallow calls of the form "Call P(...,P)".[1] Secondly, we show that it is possible to obtain a sound and complete system of Hoare-like axioms (using static scope and allowing global variables) if we either (a) allow recursive procedures with variable parameters (call by simple name) but disallow procedure parameters or (b) allow procedure parameters but require that procedures be non-recursive.

An independent source of incompleteness is the coroutine construct. If procedures are not recursive, there is a simple method for proving correctness of coroutines, based on the addition of auxillary variables [OW76]. If, however, procedures are recursive, we show that no such simple method can give completenss. These observations generalize to languages with parallelism and recursion.

Incompleteness results can also be obtained for (a) call by name parameter passing with functions and global variables and (b) label variables with retention. All such features are too complicated for a simple axiomatic description of the type advocated by Hoare and thus in a sense inherently difficult to prove correct.

---

[1] Calls of the form "Call P(...,P)" appear to be necessary if one wants to directly simulate the lambda calculus by passing procedure parameters.

1.2 Outline of Thesis

In Chapter 2 we introduce the notion of a predicate transformer. This concept, originally due to Dijkstra, serves as an important theoretical tool throughout the thesis. The chapter begins with an informal account of Dijkstra's ideas. Next a formal syntax and semantics is given for the programming language which serves as the subject of study in the first part of this thesis. This language is basically the langauge of parameterless recursion schemes; it is the simplest programming language which illustrates all of the difficulties associated with recursion. The additional problems caused by block structure and various parameter passing mechanisms are considered in Chapter 5. Once the formal semantics for the programming language has been developed, we show how Dijkstra's ideas on predicate transformers can be made rigorous. Several generalizations of Dijkstra's original ideas and a proof of his fundamental invariance theorem are also given. We conclude the chapter with the predicate transformer fixed point theorems; this is one of the main technical results of the thesis. The theorem is used in later chapters to obtain proofs of soundness and (relative) completeness for various control structures.

Chapter 3 is a discussion of proof systems for partial correctness. We introduce Cook's notion of expressibility and discuss how his completeness results for Hoare-like axiom systems are related to the results of deBakker and Meertens [DE73]. The chapter is concluded by showing how the predicate transformer fixed point theorem may be used to obtain proofs of soundness and completeness for Hoare-like axiom systems.

In Chapter 4 the problems posed by non-regular control structures are considered. We show how these difficulties are handled by deBakker and Meertens [DE73] and also by Gorelick [GO75]. Finally, we show how the proof system of the preceeding chapter can be extended to handle total correctness.

In the final chapter we consider more complicated programming language constructs and ask the question of whether there are natural constructs for which it is impossible to obtain sound and complete systems of Hoare-like axioms. We show that it is impossible to obtain good axioms for recursive procedures with procedure parameters if static scope is used and global variables are allowed. Various ways of restricting the programming language so that it is possible to obtain sound and complete systems of axioms are examined. The problem of obtaining a sound and complete system of axioms for coroutines is also considered. We conclude by mentioning additional applications of these ideas and by discussing the possible impact of these results on future developments in programming languages.

CHAPTER 2

A FIXED-POINT CHARACTERIZATION OF PARTIAL AND TOTAL CORRECTNESS

2.1 Predicate Transformers

In his paper entitled "A Simple Axiomatic Basis for Programming Language Constructs," E.W. Dijkstra describes a method of proving total correctness of computer programs. Dijkstra's method is notable because of its simplicity and the light that it sheds on related approaches to program correctness such as the inductive assertions method of Floyd and the axiomatic approach of Hoare. In this chapter we describe Dijkstra's method and show how it can be formalized within the framework of a least fixed point theory of programming language semantics. We also indicate some directions in which Dijkstra's work can be extended. In particular, we show that Dijkstra's ideas can be used to simplify the task of proving soundness and (relative) completeness of Hoare-like proof systems for partial correctness. Thus, we regard Dijkstra's work more as a tool for the study of partial and total correctness rather than as an independent system for constructing proofs.

Dijkstra specifies the meaning of programs by giving a set of rules for obtaining the weakest precondition which should be satisfied by the initial state of the program in order to guarantee that the program will terminate and that a given post condition will be satisfied by the final state of the program.

For example, assume that we are given a program A which supposedly computes the square root of its input variable x and assigns it to the variable y. In this example, the post condition P is obviously given

by $y = \sqrt{x}$. If the weakest precondition corresponding to P turned out to be $x \geq 5$ then we would know that in order for our alogorithm to correctly compute the square root of x, x would have to be greater than 5 and that if x were less than 5 we could expect to get erroneous results. Faced with such a result we would probably refuse to approve A as a square root computing routine and continue searching for a different program with post condition P and the correct weakest precondition of $x \geq 0$.

More formally, we regard the weakest precondition as a function of the program A and the post condition P and we denote this function by BA(P).[1] It should be clear that as far as a total correctness is concerned the predicate transformer BA(P) completely describes the meaning of the program A. What we desire is a specification of this predicate transformer for the atomic statements of the programming language and a set of rules for deriving the predicate transformer for a composite program from the predicate transformers of its component parts.

For certain types of statements these rules are quite obvious:

(1) If A is the null statement then obviously we want BA(P) = P for all predicates P.

(2) If A is the assignment statement x: = e where e is some expression, then BA(P) should be $P\frac{e}{x}$ (we assume that the predicates are expressed as formulas in some logical system such as predicate calculus: $P\frac{e}{x}$ means the predicate P with all occurrences of the variable x replaced by the expression e). For example, if A is the statement "x := x + 1" and P is the predicate $\left(x + y = 20 \text{ and } x < 9\right)$ then BA(P) is the predicate

[1] Dijkstra uses the term predicate transformer for any function which maps a statement - predicate pair into another predicate.

$\{(x + 1) + y = 20 \text{ and } (x + 1) < 9\}$. The reader should note the resemblence of this rule to Hoare's axiom for the assignment statement.

(3) If A is the composition of two statements A_1 and A_2 i.e. $A = (A_1; A_2)$, then BA(P) should be $BA_1(BA_2(P))$ where BA_1 is the predicate transformer corresponding to A_1 and BA_2 is the predicate transformer for A_2.

(4) If A is a conditional statement of the form "IF b THEN A_1 ELSE A_2" then BA(P) is the predicate $[b \wedge BA_1(P)] \vee [\sim b \wedge BA_2(P)]$. Again the reader should note the resemblence to Hoare's rule for the conditional.

All of the predicate transformers defined by rules (1) to (4) have the following properties:

(i) $P = Q$ implies that $BA(P) = BA(Q)$: this condition must be satisfied if $BA(\cdot)$ is to be a function.

(ii) BA(false) = false: Dijkstra calls this property "the law of the excluded miracle." Its justification is immediate since we are dealing with total correctness.

(iii) $BA(P \wedge Q) = BA(P) \wedge BA(Q)$: this condition merely states that the set of initial states which get mapped by A onto a state in $P \wedge Q$ is the intersection of the set of states which gets mapped into P and the set mapped into Q.

(iv) $BA(P \vee Q) = BA(P) \vee BA(Q)$: the justification of condition (iv) is similar to the justification of (iii).

Dijkstra calls any predicate transformer which satisfies (i)-(iv) a healthy predicate transformer.

Unfortunately, with the types of statements described above we can construct only very simple examples of programs--programs which do not

involve any repetitive constructs. It is obvious that we could considerably enrich this simple language if we allowed recursive procedures. Dijkstra does this in his paper, but his treatment of recursion is awkward and in some cases leads him to false conclusions (a counter-example to his fundamental invariance theorem for recursive procedures is given in [MC73]).

In this chapter, we combine the techniques of deBakker [DE75], Cook [CO75], and McGowan and Misra [MC73] to show how recursion can be handled correctly. We also examine several generalizations of Dijkstra's original idea. We consider predicate transformers for partial correctness and predicate transformers which give the strongest post condition corresponding to a given precondition. Finally we prove several theorems which characterize these variants on Dijkstra's original idea as the maximal and minimal fixed points of certain natural predicate transformer functionals. This is the main technical results of the Chapter. In the remaining sections we give a number of applications of this result.

2.2 Syntax and Semantics of a Simple Programming Language

In this section we describe a programming language consisting of assignment statements, conditional statements, and a simple type of parameterless procedure. Following Cook [CO75] we distinguish two formal systems involved in discussions of the correctness of computer programs: (a) the expression language L_E which is used in forming the numeric and boolean expressions of programming language itself, and (b) the assertion language L_A which is an extension of L_E and is used to describe the conditions which must hold before and after various statements of the programming language are executed. Both L_E and L_A are first order

languages (first order languages with equality) and therefore have the general structure described below:

⟨variable⟩ ::= u|v|w|u'|v'|w'|...

⟨term⟩ ::= ⟨variable⟩ | $f^{(k)}$(⟨$term_1$⟩...⟨$term_k$⟩)

where $f^{(k)}$ is a k-ary function symbol, $k \geq 0$.

⟨atomic formula⟩ ::= (⟨term⟩ = ⟨term⟩)

| $p^{(k)}$(⟨$term_1$⟩...⟨$term_k$⟩)

where $p^{(k)}$ is a K-ary predicate symbol, $k \geq 0$.

⟨formula⟩ ::= ⟨atomic formula⟩

| ~ ⟨formula⟩

| (⟨formula⟩ ∨ ⟨formula⟩)

| (⟨formula⟩ ∧ ⟨formula⟩)

| (⟨formula⟩ → ⟨formula⟩)

| ∀ u_i ⟨formula⟩

| ∃ u_i ⟨formula⟩

where u_i is some variable.

Having defined the expression and assertion languages, we can now describe the syntax of the programming language $PL[L_E, L_A]$ which we wish to study. A program is a pair ⟨H,A⟩ where H is a set of procedure declarations and A is a statement. The syntax of procedure declarations and statements is given in BNF below:

⟨Declaration⟩ ::= ⟨Procedure name⟩ ≡ ⟨Procedure body⟩

⟨Procedure body⟩ ::= ⟨Statement⟩

⟨Statement⟩ ::= ⟨Compound statement⟩

| ⟨Assignment statement⟩

| ⟨Conditional statement⟩

| ⟨Procedure call⟩

| ⟨Null statement⟩

| ⟨ Undefined statement ⟩

⟨Compound statement⟩ ::= (⟨Statement⟩ ; ⟨Statement⟩)

⟨Assignment statement⟩ ::= ⟨program identifier⟩ := ⟨numerical expression⟩

⟨Program Identifier⟩ ::= any variable of L_E

⟨Numerical expression⟩ ::= any term of L_E

⟨Conditional statement⟩ ::= (⟨boolean expression⟩ → ⟨statement⟩ , ⟨statement⟩)

⟨Boolean expression⟩ ::= any quantifier free formula of L_E

⟨Procedure call⟩ ::= ⟨Procedure name⟩

⟨Procedure name⟩ ::= X|Y|Z|X'|Y'|Z'|...

⟨Null statement⟩ ::= I

⟨Undefined statement⟩ ::= Ω

Thus, for example, the pair ⟨H,A⟩ is a typical PL program when A is the statement

((v := b(u,v); X); (u := g(u); Y))

and H contains the procedure declaration

X ≡ (b(u,v) → (u := f(u,v); X), I)

Y ≡ (v := g(u); (c(v) → (X;Y), Ω))

The semantics of the programming language $PL[L_E,L_A]$ is equally simple. An Interpretation $\mathfrak{A}$ for the programming language $PL[L_E,L_A]$ consists of

(a) a nonempty set D called the domain of the interpretation.

(b) for each k-ary function symbol $f^{(k)}$ of L_A, a k-ary function $\mathfrak{A}[f^{(k)}]$ from D to D.

(c) for each k-ary predicate symbol $P^{(k)}$ of L_A, a k-ary predicate $\mathfrak{A}[p^{(k)}]$ on the set D.

Note that since the language L_A is an extension of the language L_E, $\mathfrak{A}$

also provides assignments for the function and predicate symbols of L_E.

As is customary in treatments of first order predicate calculus, we let $L_A(\mathfrak{A})$ be the extension of L_A in which a constant symbol (0-ary function symbol) is added for each element of the domain of $\mathfrak{A}$. We then assign to each variable-free term t of L_A an element $\mathfrak{A}[t]$ of D and to each closed formula p of L_A a truth value $\mathfrak{A}[P] \in \{true, false\}$. The details of this assignment process can be found in any textbook on mathematical logic and are therefore omitted here (see, for example Schoenfield, [SC67]).

Having indicated how meanings are assigned to the variable free terms and closed formulas of L_A we can begin to discuss how meanings are assigned to programs. Let ID be the set of identifiers of the language $PL[L_E,L_A]$ (i.e. variables of L_A), then the set S of program states consists of the mappings $s : ID \to D$. If t is a term of LA with free variables $x_1,\ldots,x_n$, then

$$t(s) = t\,\frac{s(x_1)\ldots s(x_n)}{x_1\ldots x_n}$$

If P is a formula of L_A, we define $P(S)$ in an analogous manner.

The meaning of a statement A can only be described once an interpretation $\mathfrak{A}$ has been specified. Since the statement A may contain procedure calls, the meaning of A also depends on the set H of procedure declarations made in the program in which A occurs. Relative to a particular interpretation $\mathfrak{A}$ and set of procedure declarations H, the meaning of A is a function

$$M_{\mathfrak{A},H}[A] : \hat{S} \to \hat{S}$$

($\hat{S} = S \cup \{\perp\}$, where $\perp$ represents undefined)

which, intuitively speaking, gives the effect of the execution of A on

the values of the identifiers occuring in A. There are many ways that $M = M_{\mathfrak{a},H}$ can be defined--in terms of computation sequences as in [CO75] or as the least fixed point of a continuous functional as in [DE75]. We will merely list a number of properties that M can be shown to satisfy regardless of the definition used.

2.2.1 Proposition:

(a) $M[\Omega](s) = \perp$

(b) $M[I](s) = s$

(c) $M[(A_1;A_2)](s) = M[A_2](M[A_1](s))$

(d) $M[b \to A_1,A_2)](s) = \begin{cases} M[A_1](s) & \text{if } \mathfrak{a}[b(s)] = \text{true} \\ M[A_2](s) & \text{if } \mathfrak{a}[b(s)] = \text{false.} \end{cases}$

(e) $M[u := t](s) = s'$ where

$$s'(i) = \begin{cases} s(i) & \text{if } i \neq u \\ \mathfrak{a}[t(s)] & \text{if } i = u \end{cases}$$

(f) $M[X](s) = M[\tau](s)$ where X is a procedure name and the declaration $X \equiv \tau$ occurs in the set H of procedure declarations.

2.2.2 Proposition:

$M[((A_1;A_2);A_3)](s) = M[(A_1;(A_2;A_3))](s)$ for all statements A_1, A_2 and A_3 of ST and for all states $s \in S$.

Let PF be the set of partial functions from S to S, i.e. PF is the set of functions $f : \hat{S} \to \hat{S}$ such that $f(\perp) = \perp$. If $f_1, f_2 \in PF$ we write $f_1 \sqsubseteq f_2$ iff for all x, $f_1(x) = \perp$ or $f_1(x) = f_2(x)$. PF with the relation $\sqsubseteq$ is a complete partially ordered set or C.P.O. for short.

2.2.3 Proposition:

If $M[A_1] \sqsubseteq M[A_2]$ then $M[B\ \frac{A_1}{X}] \sqsubseteq M[B\ \frac{A_2}{X}]$ where A_1, A_2 and B are arbitrary statements and X is a procedure name.

The next proposition forms the basis of our treatment of recursion.

2.2.4 Proposition:

Let X be a procedure name occuring in a program which contains the procedure declaration $X \equiv \tau$. Let the statement sequence $\{X^i\}_{i\geq 0}$ be defined inductively by $X^0 = \Omega$.

$$X^{i+1} = \tau\ \frac{X^i}{X}$$

then

(a) the sequence $\{M[X^i]\}_{i\geq 0}$ is a chain in the C.P.O. PF of partial function from S to S (i.e. for all i, $M[X^i] \sqsubseteq M[X^{i+1}]$).

(b) $M[X] = \bigsqcup_{i\geq 0} M[X^i]$.

(If $\{f_i\}_{i\geq 0}$ is a chain of partial functions then $h = \bigsqcup_{i\geq 0} f_i$ may be defined by: $h(x) = y$ if there exists a $j \geq 0$ such that $f_i(x) = y$ for $i \geq j$. $h(x) = \perp$ otherwise.)

We are now ready to discuss what it means for a program to be correct.

2.2.5 Definition:

Let A be a statement of $PL[L_E, L_A]$. Let P and Q be formulas of the assertion language L_A. We say that A is partially correct with respect to the precondition P and post condition Q (and relative to some interpretation $\mathfrak{a}$ and set of procedure declarations H) iff

$$\forall s, s' \in S\ [(\mathfrak{a}[P(s)] = \text{true})$$
$$\wedge\ (M[A]\ (s) = s')$$
$$\rightarrow\ (\mathfrak{a}[Q(s')] = \text{true})]$$

If A is partially correct with respect to precondition P and post condition Q then we write $\models_{\mathfrak{a}} \{P\}\ A\ \{Q\}$ or $\models \{P\}\ A\ \{Q\}$ if the interpretation $\mathfrak{a}$ is clear.

Note that a statement A may be partially correct with respect to the precondition P and post condition Q and still not terminate for some initial state s which satisfies the precondition P. Since, in addition, we may wish to require that the statement terminate for any state s which satisfies the precondition P, we will introduce another notion of correctness called total correctness.

2.2.6 Definition:

Let A, P, and Q be as in the preceding definition. We say that A is totally correct with respect to the precondition P and post condition Q and relative to some interpretation $\mathfrak{a}$

iff

$$\forall s \in S\ [(\mathfrak{a}[P(s)] = \text{true}) \rightarrow$$
$$\exists s' \in S\ [(M[A]\ (s) = s')$$
$$\wedge\ (\mathfrak{a}[Q(s')] = \text{true})]]$$

If A is totally correct with respect to precondition P and post condition Q then we will write $\models_{\mathfrak{a}} \langle P \rangle\ A\ \langle Q \rangle$.

We conclude this section by considering a simple example. Suppose that a program contains the procedure declaration

$$X \equiv (b(u,v) \rightarrow (u := f(u,v);\ X),\ I)$$

in its declaration part. If $\mathfrak{a}$ is the interpretation in which $D = N$, b is assigned the predicate $b'(x,y) = \{x \geq y\}$ and f is assigned the function $f'(x,y) = x - y$, then we can view the procedure X as yielding:

$$X \equiv (u \geq v \rightarrow (u := u - v;\ X),\ I)$$

Thus, if $\mathfrak{a}$ also assigns the constant a the value $0 \in D$, we have

$$\vdash_{\mathfrak{a}} \{b(u,a) \wedge b(v,a)\}\ X\ \{\sim b(u,v)\}$$

i.e. the procedure call X is partially correct with respect to the precondition $P \equiv u \geq 0 \wedge v \geq 0$ and post condition $Q \equiv u < v$. Note that X is not totally correct with respect to P and Q since in executing a call on the procedure X if $u = 0$ and $v = 0$ initially, P will be satisfied but the procedure will diverge.

2.3 Healthy Predicate Transformers Revisited

We now indicate how Dijkstra's method is proving program correctness can be formalized within the theoretical framework given in section 2.2 and show that a satisfactory treatment of recursion can be given. For similar developments of Dijkstra's method, the reader should see deBakker [DE75] or McGowan and Misra [MC73].

Before we can talk about predicate transformers we must agree on a method of representing predicates. We assume that an interpretation $\mathfrak{a}$ has been given for the programming language $PL[L_E, L_A]$ and that the set H of procedure declarations has been fixed. S will denote the set of possible machine states. We shall temporarily disregard

expressibility considerations and make the convention that a predicate is merely a subset of the state set S and that the set of all predicates is just the power set of S, $P(S)$. (Thus for the moment we ignore the fact that if S is infinite there will be many subsets of S which do not have recursive descriptions). If the convention above is followed then logical operations on predicates can be interpreted as set theoretic operations on subsets of S i.e. "or" becomes "union", and "and" becomes "intersection", "not" becomes "complement", and "implies" becomes "is a subset of ".

Under the conventions made above, we shall see that if A is a statement of ST and $P \subseteq S$ is a predicate then it's natural to define the weakest precondition corresponding to A and P to be the set $\{s \in S \mid M[A](s) \in P\}$. Formally we make the following definition:

2.3.1 Definition:

Let $B : ST \times P(S) \to P(S)$ be the predicate transformer defined by

$$B[A](P) = \{s \in S \mid M[A](s) \in P\}$$

We can immediately show that the predicate transformer $B : ST \times P(S) \to P(S)$ satisfies the four conditions necessary to be a healthy predicate transformer.

2.3.2 Proposition:

Let A be a statement of ST and let $P,Q \subseteq S$ be predicates, then

(a) $P \subseteq Q$ implies $B[A](P) \subseteq B[A](Q)$

(b) $B[A]$ (false) = false where false $= \emptyset \subseteq S$

(c) $B[A](P \wedge Q) = B[A](P) \wedge B[A](Q)$

(d) $B[A](P \vee Q) = B[A](P) \vee B[A](Q)$

Proof: Let $g = M[A]$ so that $g : S \to S$ then $B[A](P) = g^{-1}(P)$ and each of the four properties above follows from a property of inverse images of sets. For example (b) is true because $g^{-1}(\emptyset) = \emptyset$ and (d) is true because $g^{-1}(P \cup Q) = g^{-1}(P) \cup g^{-1}(Q)$. In fact (c) and (d) can be strengthened to apply to families of predicates.

2.3.3 Proposition:

Let A be a statement of ST and let $\{P_i\}_{i \geq 0}$ be a family of predicates on S then

(a) $B[A](\bigcup_{i \geq 0} P_i) = \bigcup_{i \geq 0} B[A](P_i)$

(b) $B[A](\bigcap_{i \geq 0} P_i) = \bigcap_{i \geq 0} B[A](P_i)$

In much the same manner it is possible to handle the null statement, the undefined statement and the composition to two statements.

2.3.4 Proposition:

(a) $B[I](P) = P$

(b) $B[\Omega](P) = \text{False}$

(c) $B[(S_1;S_2)](P) = B[S_1](B[S_2](P))$

Proof of (c): $s \in B[(S_1;S_2)](P)$ iff $M[(S_1;S_2)](s) = s'$ and $s' \in P$. But $m[(S_1;S_2)](s) = s'$ iff $\exists\, s'' \ni M[S_1](s) = s''$ and $M[S_2](s'') = s'$. Thus $s \in B[(S_1,S_2)](P)$ iff $M[S_1](s) = s''$ and

$s'' \in B[S_2](P)$. Repeating the argument used in the preceding line we obtain the desired result that $s \in B[(S_1;S_2)](P)$ iff $s \in B[S_1](B[S_2](P))$.

The treatment of recursion is slightly more complicated but uses the same basic idea.

2.3.5 Proposition:

Let X be a procedure name occuring in a program which contains the procedure declaration $X \equiv \tau$. Let $P \subseteq S$ be a predicate. Define the statement sequence $\{X^i\}_{i\geq 0}$ by induction:

$$X^0 = \Omega$$
$$X^{i+1} = \tau\frac{X^i}{X}$$

then

(a) The sequence $\{B[X^i](P)\}_{i\geq 0}$ is an ascending chain in $P(S)$, i.e. for $i \geq 0$, $B[X^i](P) \subseteq B[X^{i+1}](P)$

(b) $B[X](P) = \bigcup_{i\geq 0} B[X^i](P)$

Proof of (a): $s \in B[X^i](P)$ implies that there is an s' such that $M[X^i](s) = s'$ and $s' \in P$. But by proposition 2.2.4 this implies that $M[X^{i+1}](s) = s'$ and $s' \in P$.
Thus $s \in B[X^{i+1}](P)$.

Proof of (b): We show first that $B[X](P) \subseteq \bigcup_{i\geq 0} B[X^i](P)$.
Let $s \in B[X](P)$. Then there is an $s' \in S$ such that $M[X](s) = s'$ and $s' \in P$. But $M[X](s) = \bigsqcup_{i\geq 0} M[X^i](s)$. Hence there must be a $j \geq 0$ such that $M[X^j](s) = s'$. Since $s' \in P$ it follows that $s \in B[X^j](P)$ and hence that $B[X](P) \subseteq \bigcup_{i\geq 0} B[X^i](P)$. The containment $\bigcup_{i\geq 0} B[X^i](P) \subseteq$

B[X](P) follows since each step in the preceding argument is reversible.

In order to propertly treat the assignment statement and the conditional statement we need to introduce a simple notion of expressibility.

2.3.6 Definition:

Let p be a formula of the assertion language L_A. We say that p expresses the predicate $P \subseteq S$ iff for all $s \in S$ $\mathfrak{a}[p(s)] = \text{true} \Leftrightarrow s \in P$.

2.3.7 Proposition:

Let p be a formula and let $P \subseteq S$ be a predicate such that p expresses P. Let A be the assignment statement u := t where t is a term of L_E, then $p\ \frac{t}{u}$ expresses the predicate B[A](P).

Proof: It is easy to show that if p is an arbitrary formula of $L_A(\mathfrak{a})$ and t is an arbitrary term of $L_A(\mathfrak{a})$ then

$$\mathfrak{a}[p\ \tfrac{t}{u}\ (s)] = \mathfrak{a}[p\ \tfrac{\mathfrak{a}[t(s)]}{u}\ (s)]$$

In order to prove the proposition we must show that if p expresses P then $p\ \frac{t}{u}$ expresses B[A](P). This is equivalent to showing that $\mathfrak{a}[p\ \frac{t}{u}\ (s)]$ is true iff $s' = M[u := t](s)$ and $s' \in P$. But $s' = M[u := t](s)$ iff

$$s'(i) = s(i) \quad i \neq u$$

and

$$s'(u) = \mathfrak{a}[t(s)]$$

Since, by hypothesis, p expresses P we have that $s' \in P$ iff $\mathfrak{a}[p(s')]$ is true. It follows that $s' = M[u := t](s)$ and $s' \in P$ iff $\mathfrak{a}[p\ \frac{\mathfrak{a}[t(s)]}{u}\ (s)]$ is true. Hence the desired result follows from the identity stated at the beginning of the proof.

Similarly, we have the following rule for the conditional.

2.3.8 Proposition:

Let the formula q express the predicate $Q \subseteq S$. Then $B[q \rightarrow A_1, A_2](P) = (Q \wedge B[A_1](P)) \vee (\sim Q \wedge B[A_2](P))$.

Proof: $s \in B[q \rightarrow A_1, A_2](P)$ iff $M[q \rightarrow A_1, A_2](s) = s'$ and $s' \in P$. But $M[q \rightarrow A_1, A_2](s) = s'$ iff $\mathfrak{a}[q(s)] = \text{true}$ (i.e. $s \in Q$) and $M[A_1](s) = s'$ or $\mathfrak{a}[(s)] = \text{false}$ (i.e. $s \in \sim Q$) and $M[A_2](s) = s'$. It follows immediately that $s \in B[q \rightarrow A_1, A_2](P)$ iff $s \in [Q \wedge B[A_1](P)) \vee (\sim Q \wedge B[A_2](P))$.

Henceforth, if the formula p expresses the predicate P then (by a convenient abuse of notation) we will use p and P interchangeably in predicate transformer equations. We must always remember, however, that there may be many $P \subseteq S$ for which there does not exist a formula p such that p expresses P.

We illustrate how the above properties of the predicate transformer B can be used to compute $B[A](P)$. Let L_A be the language L_n of number theory and let $\mathfrak{a}$ be an interpretation in which the symbols of L_n get their standard meanings. Suppose that $A = (A_1; A_2)$ where A_1 is the assignment $u := u-v$ and A_2 is the conditional $(u \geq 2 \rightarrow v := 3, v := u-4)$. If P is the formula $u + v \leq 6$ then $B[A_2](P) = (u \geq 2 \wedge u + 3 \leq 6) \vee (u < 2 \wedge u + (u - 4) \leq 6) = (u \leq 3)$. Hence $B[A](P) = B[A_1](B[A_2](P)) = \{u-v \leq 3\}$

We have now shown that $B[A](P)$ has all of the properties that we argued in 2.1 a healthy predicate transformer should have. We next investigate the relation between the predicate transformer B and the notion of total correctness. Since, by the abuse of notation mentioned above,

we are identifying the formula p and the predicate $\{s \mid \mathfrak{a}[p(s)] = \text{true}\} \subseteq S$. The definitions of partial and total correctness now simplify to:

partial correctness:

$$\{p\}\ A\ \{q\} \text{ iff}$$

$$\forall s, s' \in S\ [s \in p \wedge M[A](s) = s' \Rightarrow s' \in q]$$

total correctness:

$$\langle p\rangle\ A\ \langle q\rangle \text{ iff}$$

$$\forall s[s \in p \Rightarrow \exists s'\ [M[A](s) = s'\ s' \in q]]$$

<u>2.3.9 Proposition (total correctness):</u>

Let A be a statement of $PL[L_E, L_A]$ and let p and q be formulas of the assertion language L_A then $\models \langle p\rangle\ A\ \langle q\rangle$ iff $p \subseteq B\ [A](q)$.

<u>Proof</u>: ($\Rightarrow$) assume that $\langle p\rangle\ A\ \langle q\rangle$ is true. Then $\forall s \in S\ [s \in p \Rightarrow \exists s'\ [M[A](s) = s' \wedge s' \in q]]$. Let $s \in p$. By the definition of total correctness above we see that there must exist an s' such that $s' = M[A](s)$ and $s' \in q$. But this means that $s \in B[A](q)$. Thus $p \subseteq B[A](q)$ ($\Leftarrow$) assume that $p \subseteq B[A](q)$. Let $s \in p$. Then $s \in B[A](q)$. Hence $M[A](s) = s'$ and $s' \in q$ thus $\forall s \in S[s \in P \Rightarrow \exists s'\ [M[A](s) = s'$ and $s' \in q]]$.

The above results show that we are justified in thinking of $B[A](q)$ as the weakest or most general precondition which should be satisfied by the initial state of the program A in order to guarantee that A will terminate and that q will be satisfied by the final state.

It is also possible to formulate partial correctness in terms of the predicate transformer B.

2.3.10 Proposition:

Let A, p, and q be as in the preceding proposition, then $\vdash \{p\}\ A\ \{q\}$ iff $p \wedge BA(true) \rightarrow BA(q)$ where true is the identically true predicate (i.e. the entire set S).

Proof: Similar to the proof given on the preceding page.

By way of example note that if X is the procedure defined at the end of section 2.2 then $B[X](true) = B[X](\{u < v\}) = \{v > 0\}$. Thus, by proposition 2.3.9, X is totally correct with respect to precondition $\{v > 0\}$ and post condition $\{u < v\}$. By proposition 2.3.10 we see that X is partially correct with respect to precondition $P = \{v \geq 0\}$ and post-condition $Q = \{u < v\}$ since $P \wedge B[X](true) \rightarrow B[X](Q)$ is true,

We will see however that it is easier to treat partial correctness using the two generalizations of Dijkstra's orginal idea which are discussed in the next section.

2.4 Forward and Reverse Predicate Transformers for Partial Correctness.

In this section we introduce two generalizations of Dijkstra's original idea both of which are especially designed to handle questions relating to partial correctness. The first of the two predicate transformers to be considered is denoted by R and associates with a statement A and a post condition Q the weakest precondition R[A](Q) which must be satisfied initially in order to guarantee that the predicate Q is satisfied when the statement terminates (if it does terminate). Formally, we have the following defintion:

2.4.1 Definition:

Let $R : ST \times P(S) \rightarrow P(S)$ be defined by $R[A](Q) =$

$\{s \in S \mid M[A](s) = \perp$ or $M[A](s) \in Q\}$ (We shall call R the reverse predicate transformer for partial correctness.)

The relationship between the predicate transformer R and the predicate transformer B defined in the preceding section is immediate.

2.4.2 Proposition:

Let A be a statement in ST and let $Q \subseteq S$ be a predicate, then $R[A](Q) = \sim B[A](T) \vee B[A](P)$

Proof: B[A](T) is the set of states on which the program A terminates.

The second predicate transformer which we wish to consider is denoted by F and is in some sense the dual of the predicate transformer R defined above. F associates with a given statement A and precondition P the strongest post condition which will be satisfied by the final state of A provided that the initial state satisfies the precondition P.

2.4.3 Definition:

Let $F : ST \times P(S) \to P(S)$ be defined by $F[A](P) = \{M[A](P) \mid s \in P\}$ we will call F the forward predicate transformer for partial correctness[1]).

[1] Note that there is no interesting analogue F' of a forward predicate transformer for total correctness. This is the case because if we are dealing with total correctness and the precondition P contains a state for which the program A does not terminate then $F'[A](P)$ would have to be undefined. If on the other hand, A terminates for every state in P then we would have $F'[A](P) = F[A](P)$.

2.4.4 Proposition:

Let A be a statement of $PL[L_E, L_A]$ and let P and Q be formulas of L_A then the following are equivalent:

(1) {P} A {Q}

(2) $P \rightarrow R[A](Q)$

(3) $F[A](P) \rightarrow Q$

Proof: We show only that (1)$\Longleftrightarrow$(3) (1)$\Rightarrow$(3). Assume that {P} A {Q} is true. Then $\forall$ s, s'[s $\in$ P $\wedge$ M[A](s) = s' $\Rightarrow$ s $\in$ Q]. Let s' $\in$ F[A](P) then s' = M[A](s) and s $\in$ P. By above assumption we get s' $\in$ Q. Thus when we view F[A](P) and Q as sets we know that $F[A](P) \subseteq Q$. (3)$\Rightarrow$(1): Assume that $F[A](P) \subseteq Q$. Let s $\in$ P. and M[A](s) = s' then s' $\in$ F[A](P) so s' $\in$ Q. Thus we conclude that $\forall$ s, s' [s $\in$ P $\wedge$ M[A](s) = s' $\Rightarrow$ s' $\in$ Q] is indeed true.

Similarly, we obtain

2.4.5 Proposition:

Let A be a statement of $PL[L_E, L_A]$ and let P,Q be formulas of L_A then the following are true

(1) {R[A](Q)} A {Q}

(2) {P} A {F[A](P)}

The predicate transformer R satisfies all of the conditions necessary to be a healthy predicate transformer with the exception of the law of the excluded miracle. The law of the excluded miracle in some sense characterizes total correctness.

2.4.6 Proposition:

Let A be a statement in ST.

(a) if $P, Q \subseteq S$ are predicates then $P \subseteq Q$ implies $R[A](P) \subseteq R[A](Q)$.

(b) if $\{P_i\}_{i\geq 0}$ is a family of predicates then

$$R[A]\left(\bigcup_{i\geq 0} P_i\right) = \bigcup_{i\geq 0} R[A](P_i)$$

and

$$R[A]\left(\bigcap_{i\geq 0} P_i\right) = \bigcap_{i\geq 0} R[A](P_i)$$

Proof: Similar to the proof of proposition 2.3.2.

In a manner analogous to (b) we prove the following.

2.4.7 Proposition:

Let $Q(u)$ and $P(u)$ be formulas in L_A with free variable u (u must not be changed by the statement A) such that $P(u)$ expresses $R[A](Q(u))$, then

(a) $R[A](\exists u\ Q(u)) = \exists u\ P(u)$ and

(b) $R[A](\forall u\ Q(u)) = \forall u\ P(u)$

By a slight abuse of notation we write

(A') $R[A](\exists u\ Q(u)) = \exists u\ R[A](Q(u))$ and

(B') $R[A](\forall u\ Q(u)) = \forall u\ R[A](Q(u))$

In addition, with the exception of the rule for Ω and the rule for recursion the rules for "computing" $R[A](P)$ are the same as those for $B[A](P)$.

2.4.8 Proposition:

(1) $R[\Omega](P) = \text{true}$

(2) $R[I](P) = P$

(3) $R[(A_1;A_2)](P) = R[A_1](R[A_2](P))$

(4) $R[u := t](P) = P\frac{t}{u}$

(5) $R[b \rightarrow A_1, A_2](P) = (b \wedge R[A_1](P)) \vee (\neg b \wedge R[A_2](P))$

Proof: See the proof of propositions 2.3.4, 2.3.7, and 2.3.8.

As one would expect the rule for recursion is the dual of the one given in 2.3.5.

2.4.9 Proposition:

Let X be a procedure name occuring in a program which contains the procedure declaration $X \equiv \tau$.

Let $P \subseteq S$ be a predicate. Define the statement sequence $\{X^i\}_{i \geq 0}$ by induction

$$X^0 = \Omega$$
$$X^{i+1} = \tau\frac{X^i}{X}$$

then

(A) The sequence $\{R[X^i](P)\}_{i \geq 0}$ is a descending chain in $P(S)$, i.e. for all $i \geq 0$ $R[X^{i+1}](P) \subseteq R[X^i](P)$.

(B) $R[X](P) = \bigcap_{i \geq 0} R[X^i](P)$.

Proof of (A): Let $s \in R[X^{i+1}](P)$ then $M[X^{i+1}](s) = \perp$ or there is an s' such that $M[X^{i+1}](s) = s'$ and $s' \in P$. If $M[X^{i+1}](s) = \perp$ then $M[X^i](s) = \perp$ also. If $M[X^{i+1}](s) = s'$ then either $M[X^i](s) = \perp$ or $M[X^i](s) = s'$ and $s' \in P$. Thus in any case $s \in R[X^i](P)$.

Proof of (B): Let $s \in R[X](P)$ then either $M[X](s) = \perp$ or there is an s' such that $M[X](s) = s'$ and $s' \in P$. Thus either for all $i \geq 0$ $M[X^i](s) = \perp$ or there is a j such that $i \geq j$ implies $M[X^i](s) = s'$. But this implies that for each i either $M[X^i](s) = \perp$ or there is an s' such that $M[X^i](s) = s'$ and $s' \in P$. It follows that

$s \in R[X^i](P)$ for all i. Hence $s \in \bigcap_{i \geq 0} R[X^i](P)$ and thus $R[X](P) \subseteq \bigcap_{i \geq 0} R[X^i](P)$.

Conversely, let $s \in \bigcap_{i \geq 0} R[X^i](P)$, then for all $i \geq 0$, $s \in R[X^i](P)$. So for all $i \geq 0$, $M[X^i](s) = \perp$ or there exists an $s' \in S$ such $M[X^i](s) = s'$ and $s' \in P$. But since the sequence $M[X^i](s)$ is a chain we get that either for all $i \geq 0$ $M[X^i](s) = \perp$ or there exist s' and j such that $i \geq j$ implies that $M[X^i](s) = s'$ and $s' \in P$. Hence $M[X](s) = \perp$ or there is an s' such that $M[X](s) = s'$ and $s' \in P$. This means that $s \in R[X](P)$ so we conclude that $\bigcap_{i \geq 0} R[X^i](P) \subseteq R[X](P)$.

The predicate transformer $F: ST \times P(S) \to P(S)$ possesses a similar set of properties.

2.4.10 Proposition:

(A) If $P, Q \subseteq S$ are predicates then $P \subseteq Q$ implies $F[A](P) \subseteq F[A](Q)$.

(B) $F[A](\text{False}) = \text{False}$

(C) If $\{P_i\}_{i \geq 0}$ is a family or predicates on S

then $F[A]\left(\bigcup_{i \geq 0} P_i\right) = \bigcup_{i \geq 0} F[A](P_i)$

and $F[A]\left(\bigcap_{i \geq 0} P_i\right) \subseteq \bigcap_{i \geq 0} F[A](P_i)$

Note that (B) does not imply total correctness in the case of the forward predicate transformer. Note further that it is impossible to claim equality in the second part of (C).

Proof: The proof of this proposition is similar to the proof of proposition 2.3.2. Let $g : S \to S$ be defined by $g(s) = M[A](s)$ then properties (A), (B), and (C) follow from properties of direct images of sets under the mapping g. If we translate (C) from a satement about sets to a logical assertion we obtain:

2.4.11 Proposition:

Let $Q(u)$ and $P(u)$ be formula in L_A with free variable u (u must not be changed by the statement A) such that $P(u)$ expresses $F[A]\ (Q(u))$ then

(A) $F[A]\ (\exists u\ Q(u)) = \exists u\ P(u)$

(B) $F[A]\ (\forall u\ Q(u)) \rightarrow \forall u\ P(u)$

Again by a slight abuse of notation, we write

(A') $F[A]\ (\exists u\ Q(u)) = \exists u\ F\ [A]\ (Q(u))$

(B') $F[A]\ (\forall u\ Q(u)) \rightarrow \forall u\ F\ [A]\ (Q(u))$

The rules for "computing" $F[A]\ (P)$ are analogous to the rules for "computing" $R[A]\ (P)$.

2.4.12 Proposition:

Let $P \subseteq S$ be a predicate, then

(A) $F[\Omega]\ (P) = \text{false}$

(B) $F[I]\ (P) = P$

(C) $F[(A_1;A_2)]\ (P) = F[A_2]\ (F[A_1]\ (P))$

(D) If X and the sequence $\{X^i\}_{i\geq 0}$ are as in proposition 2.4.9 then

(i) the sequence $\{F[X^i]\ (P)\}_{i\geq 0}$ is an ascending chain in $P(S)$

(ii) $F[X]\ (P) = \bigcup_{i\geq 0} F\ [X^i]\ (P)$

(E) $F[u := t]\ (P) = \exists u'\ [u = t\ \frac{u'}{u} \wedge P\ \frac{u'}{u}\ ]$

(F) $F[(b \rightarrow A_1, A_2)]\ (P) = F[A_1]\ (P \wedge b) \vee F\ [A_2]\ (P \wedge \sim b)$

Note in (E) and (F) we are assuming that P is expressible.

Proof of E:

(i). $F[u := t]\ (P) \subseteq \exists u'\ [u = t\ \frac{u'}{u} \wedge P\ \frac{u'}{u}\ ]$

Let $s \in F\ [u := t]\ (P)$ then there exists an s' such that $M[u := t]\ (s') = s$ and $s' \in P$.

By proposition 2.2.1 we know that

$s(u) = \mathfrak{a}[t(s')]$

$s(i) = s'(i) \quad i \in Id, \quad i \neq u$

Thus, we have that

$$\mathfrak{a}[u(s)] = \mathfrak{a}[t \frac{s'(u)}{u} (s)]$$

and

$$\mathfrak{a}[P \frac{s'(u)}{u} (s)] = true$$

It follows that

$$\mathfrak{a}[(u = t \frac{s'(u)}{u} \wedge P \frac{s'(u)}{u}) (s)] = true$$

Hence

$$\mathfrak{a}[\exists\ u' (u = t \frac{u'}{u} \wedge P \frac{u'}{u}) (s)] = true$$

and

$$s \in \exists\ u' (u = t \frac{u'}{u} \wedge P \frac{u'}{u})$$

(ii). $\exists\ u'[u = t \frac{u'}{u} \wedge P \frac{u'}{u}] \subseteq F[u := t] (P)$.

Let $s \in \exists\ u' [u = t \frac{u'}{u} \wedge P \frac{u'}{u}]$

then

$$\mathfrak{a}[\exists\ u'[u = t \frac{u'}{u} \wedge P \frac{u'}{u}] (s)] = true$$

so there is an $a \in D$ such that

$$\mathfrak{a}[(u = t \frac{a}{u} \wedge P \frac{a}{u}) (s)] = true.$$

Let s': $ID \to D$ be defined by $s'(i) = \begin{cases} a & \text{if } i = u \\ s(i) & \text{o.w.} \end{cases}$

Since $\mathfrak{a}[u(s)] = \mathfrak{a}[t \frac{a}{u} (s)]$ and $\mathfrak{a}[P \frac{a}{u} (s)] = true$,

we have

$$\mathfrak{a}[u(s)] = \mathfrak{a}[t(s')]$$

and

$$\mathfrak{a}[P(s')] = true$$

Thus $M[u := t] (s') = s$ and $s' \in P$

so $s \in F[u := t] (P)$.

2.4.13 Proposition:

Let A be a statement in ST and let $P, Q \subseteq S$ be predicates, then

(1) $F[A](R[A](Q)) \subseteq Q$

(2) $P \subseteq R[A](F[A](P))$

We conclude this section with two examples.

Example (1): If A is the statement in the example following proposition 2.3.8 then $R[A](P) = B[A](P) = \{u - v \leq 3\}$. This can be seen in two different ways. First the rules for "computing" $R[A](P)$ are the same as those for computing $B[A](P)$ since A does not involve the undefined statement or recursion. Secondly, recall that $R[A](P) = \neg B[A](\text{true}) \vee B[A](P)$. Since A always terminates, $B[A](\text{true}) = \text{true}$. So $R[A](P) = B[A](P)$.

The computation of $F[A](\{u - v \leq 3\})$ is also straight forward.

$F[A_1](u - v \leq 3) = \exists u'[u = u' - v \wedge u' - v \leq 3] = \{u \leq 3\}$.

Thus $F[A](u - v \leq 3) = F[A_2](F[A_1](u - v \leq 3))$

$= F[A_2](u \leq 3)$

$= F[u := 3](u \leq 3 \wedge u \geq 2) \vee F[u := u - 4](u \leq 3 \wedge u < 2)$

$= [v = 3 \wedge 2 \leq u \leq 3] \vee \text{false} = [v = 3 \wedge 2 \leq u \leq 3]$.

thus $F[A](B[A](P)) \rightarrow P$ for $P = \{u + v \leq 6\}$

as we would expect from proposition 2.4.13.

Example 2: If X is the procedure considered at the end of section 2.2, then $R[X](\{u > v\}) = \{u \geq 0\}$ while $B[X](\{u > v\}) = \{v > 0\}$. Thus we have a simple example of a situation in which $B[X](P)$ and $R[X](P)$ differ.

2.5. The Fundamental Invariance Theorem.

In this section we examine Dijkstra's Fundamental Invariance Theorem for Recursive Procedures. We state (in the notation of the preceding sections) the version of the theorem which was given in Dijkstra's original paper.

(Dijkstra): Let X be a procedure name occuring in a program which contains the procedure declaration $X \equiv \tau$. Let $P, Q \subseteq S$ be predicates. Define the statement sequence $\{X^i\}_{i \geq 0}$ by induction:

$$X^0 = \Omega$$

$$X^{i+1} = \tau \frac{X^i}{X}$$

If for all $i \geq 0$

$$P \rightarrow B\,[X^i]\,(Q)$$

implies

$$P \rightarrow B\,[X^{i+1}]\,(Q)$$

then $P \wedge B\,[X]\,(T) \rightarrow B\,[X]\,(Q)$.

As has been pointed out numerous times (see, for example, deBakker [DE75] or McGowan and Misra [MC73]) this original version of the theorem is incorrect. To see that this is the case simply choose P = true, Q = false, and $\tau = I$ (τ is the procedure body for X, I is the null statement), then all three implications reduce to (true $\rightarrow$ false). The references listed above give a slight modification of the original version of the theorem which is correct. Instead of following this course, we rephase the theorem in terms of the predicate transformers R and F (for partial correctness) and examine some of the theorem's applications.

2.5.1 Theorem:

(Dijkstra) Let X be a procedure name occuring in a program which contains the declaration $X \equiv \tau$. Let P, $Q \subseteq S$ be predicates. Define

the statement sequence $\{X^i\}_{i\geq 0}$ by induction

$$X^0 = \Omega$$

$$X^{i+1} = \tau\frac{X^i}{X}$$

if for all $i \geq 0$

$$P \to R[X^i]\ (Q)$$

implies

$$P \to R[X^{i+1}]\ (Q)$$

then we may conclude that

$$P \to R\ [X]\ (Q)$$

Proof: We rephrase the implications in terms of set inclusions. The we assume that for all $i \geq 0$

$$P \subseteq R\ [X^i](Q)$$

implies

$$P \subseteq R\ [X^{i+1}]\ (Q)$$

Note that $R\ [X^0]\ (Q) = R\ [\Omega]\ (Q) = \text{true} = S$ so that $P \subseteq R\ [X^0]\ (Q)$. By the assumption above and induction we conclude that for all $i \geq 0$ $P \subseteq R\ [X^i]\ (Q)$. Hence $P \subseteq \bigcap_{i\geq 0} R\ [X^i]\ (Q)$. But by proposition 2.4.9 $R\ [X]\ (Q) = \bigcap_{i\geq 0} R\ [X_i]\ (Q)$ thus we conclude that $P \subseteq R\ [X]\ (Q)$ or that the implication $P \to R\ [X]\ (Q)$ is true.[1]

Before going further we give a simple illustration how the fundamental invariance theorem can be used. We prove the soundness of Hoare's axiom for the while statement, i.e. the axiom

$$\frac{\{P \wedge b\}\ A\ \{P\},\quad P \wedge \sim b \to Q}{\{P\}\ X\ \{Q\}}$$

where X is bound by the declaration $X \equiv (b \to (A;\ X),\ I)$ and A is an

[1] Note the similarity of the Fundamental Invariance Theorem to the rule of Scott Induction (see [DE75] or [SC71]).

arbitrary statement of ST. To be precise, let P, Q be two formulas of L_A, then in order to prove soundness we must show that if $P \wedge b \rightarrow R[A](P)$ and $P \wedge \sim b \rightarrow Q$ are both true (under some interpretation $\mathfrak{a}$) then $P \rightarrow R[X](Q)$ is also true (in $\mathfrak{a}$). Let the sequence X^i, $i=1,2,3,\ldots$ defined by

$$X^0 = \Omega$$

$$X^{i+1} = b \rightarrow (A; X^i), \quad \text{I}$$

Note that since $P \wedge b \rightarrow R[A](P)$ and $P \wedge \sim b \rightarrow Q$ are both true by assumption, it follows that $P \rightarrow [(b \wedge R[A](P)) \vee (\sim b \wedge Q)$ is also true . Note further that $R[X^{i+1}](Q) = (b \wedge R[A](R[X^i](Q))) \vee (\sim b \wedge Q)$. Thus, if we assume that $P \rightarrow R[X^i](Q)$ it will follow that

$R[A](P) \rightarrow R[A](R[X^i](Q))$

$b \wedge R[A](P) \rightarrow b \wedge R[A](R[X^i](Q))$

$(b \wedge R[A](P)) \vee (\sim b \wedge Q) \rightarrow [b \wedge R[A](R[X^i](Q))] \vee (\sim b \wedge Q)$

or

$P \rightarrow R[X^{i+1}](Q)$

Thus, by the version of the fundamental invariance theorem given above we have that $P \rightarrow R[X](Q)$ as was required. In a subsequent section we will show that the soundness of axioms such as those are for the while statement follows directly from the greatest fixed point characterization of $R[X](Q)$.

As the reader probably expects there is also a version of the fundamental invariance theorem which applies to the forward predicate transformer R.

2.5.2 Theorem:

Let X, τ, P, Q, and X^i be as in theorem 2.5.1 then if for all

$i \geq 0$

$$F[X^i] (P) \rightarrow Q$$

implies

$$F[X^{i+1}] (P) \rightarrow Q$$

Then we may conclude that

$$F[X] (P) \rightarrow Q$$

Proof: As in theorem 2.5.1 we assume that $F[X^i] (P) \subseteq Q$ implies $F[X^{i+1}] (P) \subseteq Q$. Observe that $F[X^0] (P) = F[\Omega] (P) = \emptyset \subseteq Q$. Hence, by assumption and induction we get that $F[X^i] (P) \subseteq Q$ for all integers $i \geq 0$. Thus $\bigcup_{i \geq 0} F[X^i] (P) \subseteq Q$. By proposition 2.4.12 we know that $F[X] (P) = \bigcup_{i \geq 0} F[X^i] (P)$ thus $F[X] (P) \subseteq Q$ or equivalently $F[X] (P) \rightarrow Q$.

The duality of the forward and reverse predicate transformers for partial correctness is further emphasized by the fact that we can equally well carry out the proof of the soundness of Hoare's axiom for the while statement by using the forward predicate transformer F. This time we must show that is $F[A] (P \wedge b) \rightarrow P$ and $P \wedge \sim b \rightarrow Q$ are both true in some interpretation $\mathfrak{a}$, then $F[X] (P) \rightarrow Q$ will also be true where $X \equiv (b \rightarrow (A; X)\ I)$. Thus we assume $F[X^i] (P) \rightarrow Q$. Since $F[A] (P \wedge b) \rightarrow P$ we have

$$F[X^i] (F[A] (P \wedge b)) \rightarrow F[X^i] (P)$$

or

$$F[X^i] (F[A] (P \wedge b)) \rightarrow Q$$

Since $P \wedge \sim b \rightarrow Q$, it follows that

$$F[X^i] (P \wedge b) \vee (P \wedge \sim b) \rightarrow Q$$

or

$$F[X^{i+1}] (P) \rightarrow Q.$$

By theorem 2.5.2 we see that $F[X] (P) \rightarrow Q$.

Note that in this case there is very little difference in the complexity

of the proof of the soundness of the while statement regardless of whether we use the predicate transformer R or the predicate transformer F. We will investigate this duality more in a later section of this paper.

2.6 Predicate Transformer Fixed Point Theorems

In this section we state and prove a number of predicate transformer fixed point theorems. These theorems are important because of the light they shed on the relationship between partial and total correctness and also because they are useful in proving soundness (relative) completeness of Hoare-like axiom systems. We begin this section with a number of definitions.

2.6.1 Definition:

A function $G: P(S) \to P(S)$ is <u>additive</u> iff whenever $\{U_i\}_{i \geq 0}$ is a family of subsets of S we have

$$G(\bigcup_{i \geq 0} U_i) = \bigcup_{i \geq 0} G(U_i)$$

$G: P(S) \to P(S)$ is <u>fully additive</u> iff whenever $\{U_i\}$ is a family of subsets of S we have both

$$G(\bigcup_{i \geq 0} U_i) = \bigcup_{i \geq 0} G(U_i)$$

and

$$G(\bigcap_{i \geq 0} U_i) = \bigcap_{i \geq 0} G(U_i).$$

2.6.2 Definition:

A set of procedure declarations

$$X_1 \equiv \tau_1$$

$$X_2 \equiv \tau_2$$

$$\vdots$$

$$X_n \equiv \tau_n$$

is self-contained iff every procedure name X occuring in $\tau_1,\ldots,\tau_n$ is one of $X_1 \ldots X_n$, and none of the τ_i contains an instance of the undefined statement Ω.

2.6.3 Definition:

A self-contained system of procedure declarations

$$X_1 \equiv \tau_1$$
$$X_2 \equiv \tau_2$$
$$\vdots$$
$$X_n \equiv \tau_n$$

possesses the uniform termination property iff for all possible initial states $s \in S$ and for all i, $1 \leq i \leq n$ we have $M[X_i](s) \neq \perp$.

We next introduce the notion of a regular system of procedure declarations. The definition is modified from deBakker [DE71] and is given in cases.

2.6.4 Definition:

Let X be a procedure name, τ_1 and τ_2 be statements. Then

A. X is regular in X.

B. If τ_1 does not contain X and τ_2 is regular in X, then $(\tau_1; \tau_2)$ is regular in X.

C. If τ_1 and τ_2 are both regular in X then $(b \rightarrow \tau_1, \tau_2)$ is regular in X.

2.6.5 Definition:

A self-contained system of procedure declarations

$$X_1 = \tau_1$$
$$X_2 = \tau_2$$
$$\vdots$$
$$X_n = \tau_n$$

is a regular system iff each procedure body τ_i is regular in all of the procedure names X_j.

Intuitively, the regular systems of procedure declarations are those systems of procedure declarations which are directly representable as flow charts. Thus, for example,

(a) $X \equiv (b \rightarrow (A_1;X), I)$

(b) $X \equiv A_1; (b \rightarrow (A_2;X), Y)$

$Y \equiv (c \rightarrow (A_3;X), A_4)$

are regular systems provided none of the statements A_1, A_2, A_3, A_4 contains a procedure call, while

(c) $X \equiv (b \rightarrow A_1; X; A_2, I)$

and

(d) $X \equiv (b \rightarrow A_1; X; Y, I)$

$Y \equiv (c \rightarrow (A_2;Y), X)$

are not regular systems.

We also need the following proposition since it enables us to work with several procedure declarations simultaneously rather than one declaration at a time.

2.6.6 Proposition:

Let $X_1 \equiv \tau_1$

$X_2 \equiv \tau_2$

$\vdots$

$X_n \equiv \tau_n$

be a self-contained system of procedure declarations. Define the statement sequence $X_1^i, X_2^i, \ldots, X_n^i$ for $i = 1,2,3,\ldots$ inductively as follows:

$X_1^0 = X_2^0 \ldots = X_n^0 = \Omega$

and

$$\begin{matrix} X_1^{i+1} \\ \vdots \\ X_n^{i+1} \end{matrix} = \begin{matrix} \tau_1 \\ \vdots \\ \tau_n \end{matrix} \frac{X_1^i, X_2^i, \ldots, X_n^i}{X_1, X_2, \ldots, X_n}$$

then

(A) $$\begin{matrix} BX_1(P) \\ \vdots \\ BX_n(P) \end{matrix} = \begin{matrix} \bigcup_{i\geq 0} BX_1^i(P) \\ \vdots \\ \bigcup_{i\geq 0} BX_n^i(P) \end{matrix}$$

(B) $$\begin{matrix} RX_1(P) \\ \vdots \\ RX_n(P) \end{matrix} = \begin{matrix} \bigcap_{i\geq 0} RX_1^i(P) \\ \vdots \\ \bigcap_{i\geq 0} RX_1^i(P) \end{matrix}$$

(C) $$\begin{matrix} FX_1(P) \\ FX_n(P) \end{matrix} = \begin{matrix} \bigcup_{i\geq 0} FX_1^i(P) \\ \bigcup_{i\geq 0} FX_1^i(P) \end{matrix}$$

Proof: See the proof of propositions 2.3.5, 2.4.9 and 2.4.12. Also see proposition 2.2.4.

Let $$* \begin{cases} X_1 \equiv \tau_1 \\ \vdots \\ X_n \equiv \tau_n \end{cases}$$ be a regular system of procedure declarations and let $P \subseteq S$ be a predicate. We associate with $*$ and P a set of n predicate transformers $G_1, G_2 \ldots G_n$ where each G_i: $P(S)^n \to P(S)$. The G_i's are defined in terms of a mapping Γ_1: $ST \times P(S) \to P(S)$ i.e. for all i $G_i(U_1, U_2, \ldots, U_n) = \Gamma_1[\tau_i](P)$. Γ_1 is defined by cases

(A) $\Gamma_1[(A_1;A_2)](Q) = \Gamma_1[A_1](\Gamma_1[A_2](Q))$

(B) $\Gamma_1[(b \to A_1, A_2)](Q) = [b \wedge \Gamma_1[A_1](Q)] \vee [\sim b \wedge \Gamma_1[A_2](Q)]$

(C) $\Gamma_1[A](Q) = R[A](Q) = B[A](Q)$ if A is

the identity statement or an assignment statement.

(D) $\Gamma_1 [X_i] (Q) = U_1$

Note that $\Gamma_1 [\tau_i] (P)$ will be well defined as long as τ_i is regular in $X_1 \ldots X_n$. If X is bound by the procedure declaration $X \equiv (b \rightarrow (A;X), I)$ then the predicate transformer G associated with X and P is

$$G(U) = [b \wedge BA(U)] \vee [\sim b \wedge P]$$

$$= [b \wedge RA(U)] \vee [\sim b \wedge P]$$

provided that A is a simple statement, i.e. A does not contain Ω or any procedure calls. If $*$ is the regular system

$$X \equiv A_1; (b \rightarrow (A_2;X), Y)$$

$$Y \equiv (c \rightarrow (A_3;X), A_4)$$

then we have

$$G_1(U,V) = BA_1 ([b \wedge BA_2(U)] \vee [\sim b \wedge V])$$

$$G_2(U,V) = [c \wedge BA_3(U)] \vee [\sim c \wedge BA_4(P)]$$

We next list some of the properties of the mappings of $G_i: P(S)^n \rightarrow P(S)$. These properties can be verified by a structural induction on the regular system (*). To conserve space we illustrate each with an example rather than include a detailed proof.

2.6.7 Proposition:

$G_i: P(S)^n \rightarrow P(S)$ is fully additive in each component.

For example, if $G(U) = [b \wedge BA(U)] \vee [\sim b \wedge P]$ then

$$G(\bigcup_{i\geq 0} U_i) = [b \wedge BA(\bigcup_{i\geq 0} U_i)] \vee [\sim b \wedge P]$$

$$= [b \wedge \bigcup_{i\geq 0} BA(U_i)] \vee [\sim b \wedge P]$$

$$= \bigcup_{i\geq 0} [(b \wedge BA(U_i)) \vee (\sim b \wedge P)]$$

$$= \bigcup_{i\geq 0} G(U_i)$$

The crucial step is of course that $BA(\bigcup_{i\geq 0} U_i) = \bigcup_{i\geq 0} BA(U_i)$ which follows from proposition 2.4.6. In exactly the same way we show that $G(\bigcap_{i\geq 0} U_i) = \bigcap_{i\geq 0} G(U_i)$

2.6.8 Proposition:

G_i: $P(S)^n \to P(S)$ is monotonic in each component.

Proof: This follows immediately from the fact that each G_i is fully additive in every component.

2.6.9 Proposition:

$$G_j(BX_1^i(P),\ldots, BX_n^i(P)) = BX_j^{i+1}(P)$$

For example, if $X \equiv (b \to (A;X), I)$ then $X^0 = \Omega$ and $X^{i+1} = (b \to (A;X^i), I)$ so that $BX^{i+1}(P) = (b \wedge BA(BX^i(P))) \vee (\sim b \wedge P)$

$$= G(BX^i(P))$$

2.6.10 Proposition:

$$G_j(RX_1^i(P),\ldots RX_n^i(P)) = RX_j^{i+1}(P)$$

See the example used to illustrate 2.6.9 above and recall that excluding the undefined statement and recursion, the "rules" computing $R[A](P)$ and $B[A](P)$ are the same.

We are now ready to state the first of a series of predicate transformer fixed point theorems.

2.6.11 Theorem:

Let $*\begin{cases} X_1 \equiv \tau_1 \\ \vdots \\ X_n \equiv \tau_n \end{cases}$ be a regular system of

procedure declarations and let $P \subseteq S$ be a predicate. Define $G_1,\ldots G_n$ as indicated above and let $G: P(S)^n \to P(S)^n$ is given by $G(\langle U_1,\ldots U_n\rangle) = \langle G_1(U_1\ldots U_n)\ldots G_n(U_1\ldots U_n)\rangle$.

Then, if "$\leq$" is the natural ordering on $P(S)$ (i.e. $\langle U_1\ldots U_n\rangle \leq \langle V_1\ldots V_n\rangle$ iff $U_i \leq V_i$ for all i, $1 \leq i \leq n$) we may conclude that with respect to "$\leq$".

(A) $\langle BX_1(P),\ldots BX_n(P)\rangle$ is the <u>least</u> fixed point of G.

(B) $\langle RX_1(P),\ldots, RX_n(P)\rangle$ is the <u>greatest fixed point</u> of G.

(C) If the system * possesses the <u>uniform termination property</u> <u>then</u> G has a unique fixed point i.e.

$$\langle BX_1(P)\ldots BX_n(P)\rangle = \langle RX_1(P)\ldots RX_n(P)\rangle$$

<u>Proofof(A)</u>: We show first that $\langle BX_1(P),\ldots, BX_n(P)\rangle$ is a fixed point of G.

$$G(\langle BX_1(P),\ldots, BX_n(P)\rangle) = G(\langle \bigcup_{i\geq 0} BX_1^i(P) \ldots \bigcup_{i\geq 0} BX_n^i(P)\rangle)$$
$$= \langle G_1(\bigcup_{i\geq 0} BX_1^i(P)\ldots \bigcup_{i\geq 0} BX_n^i(P)),\ldots G_n(\bigcup_{i\geq 0} BX_1^i(P)\ldots \bigcup_{i\geq 0} BX_n^i(P))\rangle$$
$$= \langle \bigcup_{i\geq 0} G_1(BX_1^i(P),\ldots BX_n^i(P)),\ldots, \bigcup_{i\geq 0} G_n(BX_1^i(P)\ldots BX_n^i(P))\rangle$$

by propositions 2.6.7 and 2.6.8.

$$= \langle \bigcup_{i\geq 0} BX_1^{i+1}(P),\ldots, \bigcup_{i\geq 0} BX_n^{i+1}(P)\rangle$$

by proposition 2.6.9.

$$= \langle BX_1(P),\ldots, BX_n(P)\rangle$$

Next we show that $\langle BX_1(P),\ldots BX_n(P)\rangle$ is the least fixed point of G. Suppose that $\langle U_1\ldots U_n\rangle$ is a fixed point of G then (i) $\langle BX_1^0(P),\ldots, BX_n^0(P)\rangle \leq \langle U_1\ldots U_n\rangle$ and (ii) if $\langle BX_1^i(P),\ldots, BX_n^i(P)\rangle \leq \langle U_1\ldots U_n\rangle$ then we have $G(\langle BX_1^i(P),\ldots, BX_n^i(P)\rangle) \leq G(\langle U_1\ldots U_n\rangle)$ by proposition 2.6.8.

Since $\langle BX_1^{i+1}(P),\ldots, BX_n^{i+1}(P)\rangle = G(\langle BX_1^i(P),\ldots, BX_n^i(P)\rangle)$ and

$G(\langle U_1 \ldots U_n \rangle) = \langle U_1 \ldots U_n \rangle$ we get $\langle BX_1^{i+1}(P), \ldots, BX_n^{i+1}(P) \rangle \leq \langle U_1 \ldots U_n \rangle$.

Hence by induction we conclude that for all i,

$$\langle BX_1^i(P), \ldots, BX_n^i(P) \rangle \leq \langle U_1, \ldots, U_n \rangle.$$

So,

$$\langle \bigcup_{i \geq 0} BX_1^i(P), \ldots, \bigcup_{i \geq 0} BX_n^i(P) \rangle \leq \langle U_1, \ldots, U_n \rangle$$

or

$$\langle BX_1(P), \ldots, BX_n(P) \rangle \leq \langle U_1, \ldots, U_n \rangle$$

Proof of (B): The proof is similar to the proof of A. We show first that $\langle RX_1(P), \ldots, RX_n(P) \rangle$ is a fixed point of G.

$$G(\langle RX_1(P), \ldots, RX_n(P) \rangle) = G(\langle \bigcap_{i \geq 0} RX_1^i(P) \ldots \bigcap_{i \geq 0} RX_n^i(P) \rangle)$$

$$= \langle G_1(\bigcap_{i \geq 0} RX_1^i(P) \ldots \bigcap_{i \geq 0} RX_n^i(P)), \ldots, G_n(\bigcap_{i \geq 0} RX_1^i(P) \ldots \bigcap_{i \geq 0} RX_n^i(P)) \rangle$$

$$= \langle \bigcap_{i \geq 0} G_1(RX_1^i(P) \ldots RX_n^i(P)), \ldots, \bigcap_{i \geq 0} G_n(RX_1^i(P) \ldots RX_n^i(P)) \rangle$$

$$= \langle \bigcap_{i \geq 0} RX_1^{i+1}(P), \ldots, \bigcap_{i \geq 0} RX_n^{i+1}(P) \rangle$$

$$= \langle RX_1(P), \ldots, RX_n(P) \rangle$$

We next show that $\langle RX_1(P), \ldots, RX_n(P) \rangle$ is the greatest fixed point of G. Suppose that $\langle U_1 \ldots U_n \rangle$ is a fixed point of G, then (i) $\langle U_1 \ldots U_n \rangle \leq \langle RX_1^0(P), \ldots, RX_n^0(P) \rangle$ and (ii) if $\langle U_1 \ldots U_n \rangle \leq \langle RX_1^i(P), \ldots, RX_n^i(P) \rangle$ then $G(\langle U_1 \ldots U_n \rangle) \leq G(\langle RX_1^i(P), \ldots, RX_n^i(P) \rangle)$ so $\langle U_1 \ldots U_n \rangle \leq \langle RX_1^{i+1}(P) \ldots RX_1^{i+1}(P) \rangle$.

Hence, by induction, we have that for all $i \geq 0$,

$$\langle U_1 \ldots U_n \rangle \leq \langle RX_1^i(P) \ldots RX_n^i(P) \rangle$$

thus

$$\langle U_1 \ldots U_n \rangle \leq \langle \bigcap_{i \geq 0} RX_1^i(P) \ldots \bigcap_{i \geq 0} RX_n^i(P) \rangle$$

or

$$\langle U_1 \ldots U_n \rangle \leq \langle RX_1(P), \ldots, RX_n(P) \rangle$$

Proof of (C): If the system * possesses the uniform termination property, then for all j, $1 \leq j \leq n$ we have $BX_j(\text{true}) = \text{true}$ and thus that $BX_j(P) = RX_j(P)$.

Example: If X is bound by the procedure declaration $X \equiv (b \rightarrow (A;X)\ I)$ then $G(U) = [b \wedge BA(U)] \vee [\sim b \wedge P]$.

By the above theorem we know that the least fixed point of G is $BX(P)$ and that the greatest fixed point of G is $RX(P)$ (provided of course that A is simple).

Example: If the regular system under consideration is

$$X \equiv A_1;(b \rightarrow (A_2;X),\ Y)$$

$$Y \equiv (c \rightarrow (A_3;X),\ A_4)$$

then G is given by

$$G(\langle U,V \rangle) = \langle BA_1([b \wedge BA_2(U)] \vee [\sim b \wedge V]),$$
$$[c \wedge BA_3(U)] \vee [\sim c \wedge BA_4(P)] \rangle$$

In this case the least fixed point of G is the pair $\langle BX(P), BY(P) \rangle$ and the greatest fixed point is the pair $\langle RX(P), RY(P) \rangle$ (if A_1, A_2, A_3, A_4 are all simple statements.)

The reader should notice that the arguments used to establish theorem 2.6.11 do not generalize to handle non-regular systems of procedure declarations. To see that this is the case compute $\Gamma_1[\tau](Q)$ where τ is $A_1;X;A_2$ and observe what happens to A_2. If we consider predicate transformer functionals rather than simply predicate transformers in our search for fixed point equations, the argument can be made to work.

2.6.12 Definition:

Let $H_1, H_2: P(S) \to P(S)$ then we write $H_1 \sqsubseteq H_2$ iff for all $P \subseteq S$ $H_1(P) \subseteq H_2(P)$. If $\{H_i\}_{i\geq 0}$ is a family of functions such that $H_i: P(S) \to P(S)$ then we define

$$\bigsqcap_{i\geq 0} H_i: P(S) \to P(S)$$

and

$$\bigsqcup_{i\geq 0} H_i: P(S) \to P(S)$$

by $\bigsqcap_{i\geq 0} H_i(P) = \bigcap_{i\geq 0} H_i(P)$

and

$$\bigsqcup_{i\geq 0} H_i(P) = \bigcup_{i\geq 0} H_i(P)$$

2.6.13 Definition:

Let $G: (P(S) \to P(S)) \to (P(S) \to P(S))$ then G is additive iff for every family of functions $\{H_i\}_{i\geq 0}$, $H_i: P(S) \to P(S)$ we have

$$G(\bigsqcup_{i\geq 0} H_i) = \bigsqcup_{i\geq 0} G(H_i)$$

if, in addition,

$$G(\bigsqcap_{i\geq 0} H_i) = \bigsqcap_{i\geq 0} G(H_i)$$

then we say that G is fully additive.

Let $\begin{cases} X_1 \equiv \tau_1 \\ \vdots \\ X_n \equiv \tau_n \end{cases}$ be a self contained system of

procedure declarations. Let $H_i: P(S) \to P(S)$ for $1 \leq i \leq n$ and let $P \subseteq S$, then we define $G_i[H_1 \ldots H_n](P) = \Gamma_2[\tau_i](P)$, $1 \leq i \leq n$ where $\Gamma_2: ST \times P(S) \to P(S)$ is defined by:

(A) $\Gamma_2[(A_1;A_2)](P) = \Gamma_2[A_1](\Gamma_2[A_2](P))$

(B) $\Gamma_2\,[b \to A_1, A_2]\,(P) = [b \wedge \Gamma_2[A_1]\,(P)] \vee [\sim b \wedge \Gamma_2\,[A_2]\,(P)]$

(C) $\Gamma_2\,[A]\,(P) = R\,[A]\,(P) = B\,[A]\,(P)$ if A is the null statement or an assignment statement.

(D) $\Gamma_2\,[X_i]\,(P) = H_i\,(P)$

Thus Γ_2 as defined above is identical to the Γ_1 used in the argument preceding theorem 2.6.11 with the exception of case (D). Note that we can regard G_i as a function from $(P(S) \to P(S))^n$ to $(P(S) \to P(S))$.

Example: If X is bound by the procedure declaration $X \equiv (b \to A_1; X; A_2, I)$ then $G[H]\,(P) = [b \wedge BA_1\,(H(BA_2\,(P)))] \vee [\sim b \wedge P]$.

If the declaration system is

$$\begin{cases} X \equiv (b \to A_1; X; Y, I) \\ Y \equiv (c \to A_2; Y, X) \end{cases}$$

then we have

$$G_1[H_1, H_2]\,(P) = [b \wedge BA_1\,(H_1(H_2(P)))] \vee [\sim b \wedge P]$$

and

$$G_2[H_1, H_2]\,(P) = [c \wedge BA_2\,(H_2(P))] \vee [\sim b \wedge H_1\,(P)]$$

(Note that we are again assuming that A_1 and A_2 are simple statements).

Before stating theorem 2.6.17 we list a number of properties of the functionals G_i which are needed in the proof. As in the case of theorem 2.6.11 these properties can be verified, in general, by laborious structural inductions. We content ourselves with an example illustrating each.

2.6.14 Proposition:

G_i: $(P(S) \to P(S))^n \to (P(S) \to P(S)$ is fully additive in each component.

Example: Let $G[H](P) = [b \wedge BA_1(H(BA_2(P)))] \vee [\sim b \wedge P]$ as above. Then

$$\begin{aligned} G[\bigsqcup_{i\geq 0} H_i](P) &= [b \wedge BA_1(\bigsqcup_{i\geq 0} H_i(BA_2(P)))] \vee [\sim b \wedge P] \\ &= [b \wedge BA_1(\bigcup_{i\geq 0} H_i(BA_2(P)))] \vee [\sim b \wedge P] \\ &= \bigcup_{i\geq 0} G[H_i](P) = (\bigsqcup_{i\geq 0} G[H_i])(P). \end{aligned}$$

Similarly, $G[\bigsqcap_{i\geq 0} H_i](P) = (\bigsqcap_{i\geq 0} G[H_i])(P)$.

2.6.15 Proposition:

If $L_1 \sqsubseteq M_1, \ldots, L_n \sqsubseteq M_n$ in the function space $(P(S) \to P(S))$, then

$$G_i[L_1, \ldots, L_n] \sqsubseteq G_i[M_1, \ldots, M_n]$$

i.e. G_i is monotonic in each component.

Proof: This follows immediately from full additivity.

2.6.16 Proposition:

(A) $G_j[BX_1^i, \ldots BX_n^i] = BX_j^{i+1}$

(B) $G_j[RX_n^i, \ldots, RX_n^i] = RX_j^{i+1}$

For example, if $G[H](P) = [b \wedge BA_1(H(BA_2(P)))] \vee [\sim b \wedge P]$

then $G[BX^i](P) = [b \wedge BA_1(BX^i(BA_2(P)))] \vee [\sim b \wedge P]$

$= BX^{i+1}(P)$

Similarly, $G[RX^i](P) = RX^{i+1}(P)$

We are now ready to state the second predicate transformer fixed point theorem.

2.6.17 Theorem:

Let $\begin{cases} X_1 \equiv \tau_1 \\ \vdots \\ X_n \equiv \tau_n \end{cases}$ be a self-contained system of

procedure declarations. Let $G_1,\ldots, G_n$ be the associated set of predicate transformer functionals described above. Define $G: (P(S) \to P(S))^n \to (P(S) \to P(S))^n$ by $G[H_1,\ldots, H_n] = \langle G_1[H_1,\ldots, H_n],\ldots, G_n[H_1,\ldots, H_n]\rangle$ then under the natural component-wise ordering on $(P(S) \to P(S))^n$ we have

(A) $\langle BX_1,\ldots, BX_n\rangle$ is the least fixed point of G.

(B) $\langle RX_1,\ldots, RX_n\rangle$ is the greatest fixed point of G.

(C) If $*$ possesses the uniform termination property then G has a unique fixed point i.e. $\langle BX_1 \ldots BX_n\rangle = \langle RX_1 \ldots RX_n\rangle$.

Proof: The proof is exactly the same as the proof of theorem 2.6.11, using propositions 2.6.14 through 2.6.16 instead of propositions 2.6.7 through 2.6.10.

Example: If $X \equiv (b \to A_1; X; A_2, I)$ so that $G[H](P) = [b \wedge BA_1(H(BA_2(P)))] \vee [\sim b \wedge P]$ then the least fixed point of G is the predicate transformer

BX: $P(S) \to P(S)$ and the greatest fixed point is

RX: $P(S) \to P(S)$.

Similarly if the system under consideration is

$X \equiv (b \to A_1; X; Y; I)$

$Y \equiv (c \to A_2; Y, X)$

then

$$G[H_1,H_2](P) = \langle [b \wedge BA_1(H_1(H_2(P)))] \; [\sim b \wedge P] \; [c \wedge BA_2(H_2(P))] \vee [\sim c \wedge H_1(P)]\rangle.$$

The least fixed point of G is the function pair $\langle BX,BY\rangle$; the greatest fixed point is $\langle RX,RY\rangle$.

We conclude this section by stating a fixed point theorem for the forward predicate transformer F.

Let $\begin{cases} X_1 \equiv \tau_1 \\ \vdots \\ X_n \equiv \tau_n \end{cases}$ be a self-contained system of procedure declarations. Let H_i: $P(S) \to P(S)$ for $1 \leq i \leq n$ and let $P \subseteq S$ be a predicate, then we define $G_i[H_1,\ldots, H_n]\ (P) = \Gamma_3[\tau_i]\ (P)$, $1 \leq i \leq n$. Where Γ_3: $ST \times P(S) \to P(S)$ is defined by

(A) $\Gamma_3\ [A_1;A_2)]\ (P) = \Gamma_3\ [A_2]\ (\Gamma_3\ [A_1]\ (P))$

(B) $\Gamma_3\ [b \to A_1,A_2]\ (P) = \Gamma_3\ [A_1]\ (P \wedge b) \vee \Gamma_3\ [A_2]\ (P \wedge \sim b)$

(C) $\Gamma_3\ [A]\ (P) = F\ [A]\ (P)$ if A is the null statement or an assignment statement.

(D) $\Gamma_3\ [X_i]\ (P) = H_i\ (P)$.

Note that the rules for Γ_3 reflect the fact that we are dealing with the forward predicate transformer F. As in theorem 2.6.17 we can regard G_i as a function $(P(S) \to P(S))^n$ to $(P(S) \to P(S))$.

Example: If X is bound by the procedure declaration $X \equiv (b \to A;X,I)$ then

$$G[H]\ (P) = H(F\ [A]\ (P \wedge b)) \vee [P \wedge \sim b]$$

2.6.18 Theorem:

Let $\begin{cases} X_1 \equiv \tau_1 \\ \vdots \quad \vdots \\ X_n \equiv \tau_n \end{cases}$ be a self-contained system of procedure declarations. Let $G_1,\ldots G_n$ be the associated set of (forward) predicate transformer functionals (see above). Define

$G: (P(S) \to P(S))^n \to (P(S) \to P(S))^n$ by

$$G[H_1 \ldots H_n] = \langle G_1[H_1 \ldots H_n], \ldots, G_n[H_1 \ldots H_n] \rangle$$

then under the natural component-wise partial ordering on $(P(S) \to P(S))^n$ we have that $\langle FX_1, \ldots, FX_n \rangle$ is the least fixed point of G.

Proof: Similar to the proofs of theorems 2.6.11 and 2.6.17.

Example: If G is the predicate transformer functional corresponding to $X \equiv (b \to AX, I)$ i.e.

$$G[H](P) = H(F[A](P \wedge b)) \vee (P \wedge \sim b)$$

then by theorem 2.6.18 we know that the function $FX: P(S) \to P(S)$ is the least fixed point of G. Hence $FX = G[FX]$ and if $G[H] = H$ then $FX \sqsubseteq H$ in $(P(S) \to P(S))$.

Chapter 3

Expressibility, Soundness, and Completeness

3.1 Introduction

We now wish to turn our attention to axiom systems for partial correctness of programs. In particular we will be concerned with what constitutes a "good" axiom or axiom system. A "good" axiom or rule of inference must, of course, be simple to understand and easy to use. This criterion must always involve a subjective element. More formal criteria usually involve some notion of soundness and/or completeness. In this chapter we give the usual definition of soundness [CO75] and try to relate two notions of completeness which are found in the literature on partial correctness. The first of these is the idea of an adequate proof rule and is due to J.W. deBakker [DE73 and DE71]. The second is a type of relative completeness first studied by S. Cook [CO75 or GO75]. It will turn out that adequacy and relative completeness are very similar concepts (i.e. the second is implied by the first) and that both can be demonstrated using the predicate transformer fixed point theorems studied in the last chapter.

To illustrate the concepts mentioned above and to lay the foundation for remainder of the chapter, we introduce the notion of a generalized control structure. This concept will enable us to augment the simple programming language of chapter I with more elaborate control structures. We will be able to study the soundness and completeness of proof systems which allow constructs with arbitrary recursive definitions e.g. while statements, repeat statements, and more complicated statements with non-regular definitions.

3.2 Generalized Control Structures

We wish to augment the simple programming language discussed in the preceding chapter with more elaborate control structures. This can be done in general by introducing the notion of a generalized control structure (or GCS).

A GCS is specified in an extension of the language $PL[L_E,L_A]$ described in section 2.2 We extend the BNF definition of $PL[L_E,L_A]$ to include the productions

⟨statement⟩ ::= ⟨statement variable⟩

⟨statement variable⟩ ::= $\alpha_1|\alpha_2|\alpha_3$ ...

⟨boolean expression⟩ ::= ⟨boolean variable⟩

⟨boolean variable⟩ ::= $\beta_1|\beta_2|\beta_3|\ldots$

The resulting language will be denoted by $PL^+[L_E,L_A]$.

A generalized control structure C_i consists of a tuple $\langle H_i,X_i,\beta_{i_1},\ldots,\beta_{i_m},\alpha_{i_1},\ldots\alpha_{i_n}\rangle$ where H_i is a set of procedure declarations in $PL^+[L_E,L_A]$, X_i is the name of one of the procedures defined in H_i, $\beta_{i_1},\ldots,\beta_{i_m}$ are the boolean variables occuring in the declarations in H_i, and $\alpha_{i_1},\ldots,\alpha_{i_n}$ are the statement variables occuring in the declaration part H_i. We will usually refer to $\langle H_i,X_i,\beta_{i_1},\ldots,\alpha_{i_1},\ldots\rangle$ as the control structure prototype. Thus for example, a while statement would have prototype $C_1 = \langle\{X \equiv (\beta\rightarrow\alpha;\ X,I)\},\ X,\beta,\ \alpha\ \rangle$.

Finally, we consider $PL[L_E,L_A]$ augmented with control structures $C_1,C_2,\ldots,C_k$. We first extend the grammar for $PL[L_E,L_A]$ to include for $1 \leq i \leq k$.

⟨statement⟩ ::= C_i [⟨boolean expression$_1$⟩,... , ⟨boolean expression$_m$⟩ ⟨statement$_1$⟩,...,⟨statement$_n$⟩]

if the prototype for C_i is $\langle H_i,\ X_i,\ \beta_{i_1},\ldots,\beta_{i_m},\ \alpha_{i_1},\ldots\alpha_{i_n}\rangle$.

We denote the resulting language by $PL[L_E, L_A: C_1, \ldots, C_k]$ (Note that the two extensions $PL^+[L_E, L_A]$ and $PL[L_E, L_A: C_1, \ldots, C_k]$ serve different purposes and should not be confused.)

A typical program might be

```
u := f(u,v);
C1[b1, v := g(v)];
v := h(a);
C1[b2, u := h(v)]
```

if C_1 had the while statement prototype given on the preceding page, then the above would be equivalent to

```
u := f(u,v);
while b1 do v := g(v) od;
v := h(a);
while b2 do u := h(v) od;
```

The meaning function $M = M_{\mathfrak{A},H}$ may be extended to handle the new clauses of the grammar in a straightforward manner. Thus, in the example on the preceding page

$$M[C_1\,[b_1, v := g(v)]]\,(s) = M_{\mathfrak{A},H'}[Y]\,(s)$$

where H' contains the procedure declaration $Y \equiv (b_1 \rightarrow v := g(v);\ Y,\ I)$.

3.3 Proof Systems for Partial Correctness

In this section we introduce a formal system W for proving partial correctness. The formulas in this system will consist of precondition-postcondition assertions of the form $\{P\}\ A\ \{Q\}$ where P, Q are formulas of the assertion language L_A and A is a statement of the language $PL[L_E, L_A]$ (or $PL[L_E, L_A: C_1, \ldots, C_k]$ -- see the previous section). In section 2.4 of the last chapter we gave conditions

for such an assertion to be true (relative to an interpretation a): {P} A {Q} is true iff $P \rightarrow RA(Q)$ is true or, equivalently, $FA(P) \rightarrow Q$ is true.

Axioms for W will have the form {P} A {Q} i.e. they will be formulas of W. For example, the assignment statement will be characterized by the usual axiom

$$\{Q\,\tfrac{t}{u}\}\,u := t\,\{Q\} \tag{1}$$

(By the argument of proposition 2.4.12, we know that the axiom

$$\{P\}\ u := t\ \{\exists u_o\,[u = t\,\tfrac{u_0}{u} \wedge P\,\tfrac{u_o}{u}]\}$$

would work equally well.)

Rules of inference will have the form

$$\frac{H_1, H_2, \ldots, H_n}{F}$$

where F is a formula of W and each H_i is either a formula of W, a formula of L_A, or a composite formula of the form $F_1 \vdash F_2$ where F_1 and F_2 are formulas of W (Note that the notion of truth defined above for formulas of W can be extended to composite formulas without difficulty.) We will say that a rule of inference is simple, if none of the hypothesis H_i are composite formulas. We will generally be concerned with simple rules of inference in this chapter. Typical rules of inference are the simple rules for the composition of statements, the conditional statement, and consequence given below.

$$\frac{\{P\}\ A_1\ \{Q\},\ \{Q\}\ A_2\ \{R\}}{\{P\}\ (A_1;\ A_2)\ \{Q\}} \tag{2}$$

$$\frac{\{P \wedge b\}\ A_1\ \{Q\},\ \{P \wedge \sim b\}\ A_2\ \{Q\}}{\{P\}(b \rightarrow A_1;A_2)\ \{Q\}} \tag{3}$$

$$\frac{P \rightarrow Q,\ \{Q\}\ A\ \{R\}}{\{P\}\ A\ \{R\}} \quad \text{and} \quad \frac{\{P\}\ A\ \{Q\},\ Q \rightarrow R}{\{P\}\ A\ \{R\}} \tag{4}$$

The axioms and rules of inference numbered (1) - (4) above will be referred to as the basic set.

Let T be a proof system for L_A then a proof in the system (W,T) is defined in the usual way [GO75] as a sequence of formulas of W or T each of which is either an axiom or follows from preceding formulas by a rule of inference. If F is a formula in such a sequence then we write $\vdash F$ and call F a theorem of (W,T).

We say that an axiom $\{P\}\ A\ \{Q\}$ of W is valid, if it is true under all interpretations $\mathfrak{a}$. We say that a rule of inference

$$\frac{H_1,\ldots,\ H_n}{F}$$

of W is sound if whenever $H_1,H_2,\ldots,\ H_n$ are all true in some interpretation $\mathfrak{a}$, F will also be true. A simple inductive argument shows that if each axiom of W is valid and if all of the rules of inference of W are sound then every theorem of (W,T) will indeed be true (we will not concern ourselves with the soundness of the proof system T for

L_2.) Note that the soundness of the basic set of axioms and rules of inference follows immediately from propositions in section 2.4.

In section 2.5, we showed how the soundness of the rule of inference for the while statement could be established using predicate transformer techniques. We conclude this section with a less trivial example which illustrates the importance of the predicate transformer fixed point theorems.

Consider the GCS with prototype

$$C_2 = \langle [X \equiv (b_1 \to A_1;\ Y, A_2),\quad Y \equiv (b_2 \to A_2;\ X, Y)],\ X \rangle$$

As we will see later, a "good" rule of inference for this construct is given by

$$\frac{\{P \wedge b_1\}\ A_1\ \{U\},\ \{P \wedge \sim b_1\}\ A_2\ \{Q\},\ \{U \wedge b_2\}\ A_2\ \{P\}}{\{P\}\ C_2\ [b_1, b_2;\ A_1, A_2]\ \{Q\}}$$

to prove soundness of the above axiom, we assume that

$$P \wedge b_1 \to RA_1(U) \qquad \text{(i)}$$

$$P \wedge \sim b_1 \to RA_2(Q) \qquad \text{(ii)}$$

$$U \wedge b_2 \to RA_2(P) \qquad \text{(iii)}$$

are all true and try to prove that $P \to RX(Q)$ is true.

By the first predicate transformer fixed point theorem we know that $\langle RX(Q), RY(Q) \rangle$ is the greatest fixed point (even though A_1 and A_2 may not be simple statements i.e. A_1 and A_2 may not define total functions) of the system

$$G_1(V,W) = (b_1 \wedge RA_1(W)) \vee (\sim b_1 \wedge RA_2(Q))$$

$$G_2(V,W) = (b_2 \wedge RA_2(V)) \vee (\sim b_2 \wedge W).$$

It is not difficult to show that the maximality of $\langle RX(Q), RY(Q) \rangle$

implies that it is also the least upper bound of the set

$$K = \{\langle V,W\rangle | V \subseteq G_1(V,W) \text{ and } W \subseteq G_2(V,W)\}$$

(see footnote below).

From (i), (ii), (iii) above and (iv) $U \wedge \sim b_2 \rightarrow U$ we get that

$$P \rightarrow (b_1 \wedge RA_1(U)) \vee (\sim b_1 \wedge RA_2(Q))$$

$$U \rightarrow (b_2 \wedge RA_2(P)) \vee (\sim b_2 \wedge U)$$

or that

$$P \rightarrow G_1(P,U)$$

and

$$U \rightarrow G_2(P,U)$$

are both true. Making use of the correspondence between $\subseteq$ and $\rightarrow$ we see that $\langle P,U\rangle \in K$ and hence $P \rightarrow RX(Q)$ and $U \rightarrow RX(Q)$ are both true. Since $P \rightarrow RX(Q)$ was what we were required to prove, this completes the proof of soundness.

Thus we see that soundness follows from the greatest fixed point characterization of $\langle RA(Q), RY(Q)\rangle$. A similar argument could be based on the forward predicate transformer F. In this case soundness would follow from the least fixed point characterization of the strongest post condition.

3.4 Expressibility

In his paper Axiomatic and Interpretative Semantics for an Algol Fragment [CO75], Stephen Cook points out that it may happen that the

---

Let $(L,\sqsubseteq)$ be a complete lattice and let $f: L \rightarrow L$ be a monotonic function. Then f has a greatest fixed point and fixed point is given by

$$\bigsqcup \{x | x \sqsubseteq f(x)\}$$

assertion language L_A is not powerful enough to express the invariants for loops. If, for example, L_E and L_A are both the languages of Pressburger arithmetic (i.e. the language of arithmetic without multiplication), then it is easily seen that L_A will have this problem. Cook introduces the notion of expressibility to handle this difficulty. We paraphase his definition using the terminology of Chapter 2.

3.4.1 Definition:

The language L_A is F-expressive (relative to L_E and $\mathfrak{a}$) iff for every formula P in L_A and every statement A in $PL[L_E,L_A]$ (alternatively $PL[L_E,L_A: C_1,\ldots, C_k]$) there is a formula q in L_2 such that q expresses F[A] (P).[1]

It is easily seen that the full language of number theory $L_n = \langle N, ,+,\cdot,0,1\rangle$ is F-expressive (relative to L_n and an $\mathfrak{a}$ in which the symbols of L_n get their usual meanings).

In view of the close relationship between F and the predicate transformers R and B, two other definitions of expressibility immediately come to mind.

3.4.2 Definition:

The language L_A is B-expressive (relative to L_E $\mathfrak{a}$) iff for every formula q in L_A and statement A in $PL[L_E,L_A]$ (i.e. $PL[L_E,L_A: C_1,\ldots, C_k]$) there is a formula p in L_A such that p expresses B[A] (q).

---

[1] Cook also requires that "=" be in L_E and that it receive its standard interpretation. This is unnecessary in our case since we are assuming that both L_E and L_A are first order languages with equality.

3.4.3 Definition:

The language L_A is R-expressive (relative to L_E and $\mathfrak{a}$) iff for every formula q in L_A and statement A in $PL[L_E,L_A]$ there is a formula p in L_A such that p expresses $R[A](q)$.

In this section we show that all of these definitions are really equivalent, e.g. if L_A is R-expressive then it is also F-expressive, etc.

It is easily seen that B-expressibility and R-expressibility are equivalent concepts. This follows immediately from the fact that

$$BA(Q) = RA(Q) \wedge \sim RA(\text{false})$$

and

$$RA(Q) = \sim BA(\text{true}) \vee BA(Q)$$

In order to include F-expressibility in this chain of equivalences, it is convenient to introduce two new predicates A_{PAR} and A_{TOT}.

Let A be a statement of $PL[L_E,L_A]$ and let $\bar{u} = \langle u_1,\ldots, u_n\rangle$ be the variables occuring in A. Choose $\bar{x} = \langle x_1,\ldots, x_n\rangle$ and $\bar{y} = \langle y_1,\ldots, y_n\rangle$ so that none of the variables appearing in $\bar{x}$ and $\bar{y}$ also appears in $\bar{u}$. Then define

$$A_{PAR}(\bar{x},\bar{y}) \equiv \forall \bar{u}[\bar{u} = \bar{x} \rightarrow RA(\bar{u} = \bar{y})]$$

and

$$A_{TOT}(\bar{x},\bar{y}) \equiv \forall \bar{u}[\bar{u} = \bar{x} \rightarrow BA(\bar{u} = \bar{y})]$$

Using the definitions above, it is straight forward to show that

$$BA(P) \equiv \exists \bar{y}[A_{TOT}(\bar{u},\bar{y}) \wedge P\,\frac{\bar{y}}{\bar{u}}\,]$$

and

$$RA(P) \equiv \exists \bar{y}[A_{PAR}(\bar{u},\bar{y}) \wedge P\,\frac{\bar{y}}{\bar{u}}\,]$$

Combining these observations with the relationships between $BA(P)$ and $RA(P)$ described above, we get

3.4.4 Proposition:

Let A be a statement in $PL[L_E,L_A]$ ($PL[L_E,L_A: C_1,\ldots, C_k]$) then the following are equivalent:

a) BA(q) is expressible for all formulas q in L_A

b) RA(q) is expressible for all formulas q in L_A

c) $A_{TOT}(\bar{x},\bar{y})$ is expressible

d) $A_{PAR}(\bar{x},\bar{y})$ is expressible.

It is also relatively easy to show that

$$A_{PAR}(\bar{x},\bar{y}) \equiv \forall \bar{u}\,[FA(\bar{u} = \bar{x}) \rightarrow \bar{u} = \bar{y}]$$

and that

$$FA(P) \equiv \exists \bar{x}\,[A_T(\bar{x},\bar{u}) \wedge P\,\frac{\bar{x}}{\bar{y}}]$$

If we combine these relationships and use the previous proposition, we get:

3.4.5 Proposition:

Let A be a statement in $PL[L_E,L_A]$ ($PL[L_E,L_A: C_1,\ldots, C_k]$) then the following are equivalent:

a) FA(P) is expressible for all formulas P in L_A

b) $A_{TOT}(\bar{x},\bar{y})$ is expressible

c) $A_{PAR}(\bar{x},\bar{y})$ is expressible.

Finally, from propositions (3.4.4) and (3.4.5) we get

3.4.6 Proposition:

The following are equivalent:

a) L_A if F-expressive

b) L_A is R-expressive

c) L_A is B-expressive.

---

[1] Propositions 3.4.4 and 3.4.5 above were suggested in part by John Privitera.

3.5 Completeness

Given a system (W,T) for proving partial correctness (relative to an interpretation $\mathfrak{a}$) of programs in $PL[L_E,L_A: C_1,\ldots, C_k]$, we would certainly like to know that every assertion of the form $\{P\}$ A $\{Q\}$ which is true in $\mathfrak{a}$ is provable. This, of course, is a futile wish. If W and T are axiomatizable then the set of theorems which can be proved in (W,T) is R.E. However, note that the formula $\{true\}$ A $\{false\}$ is true iff A does not halt for any initial values of its input variables. It follows from recursion theory that the set of all such true formulas may fail to be R.E. and that as a result we cannot hope for the type of completeness described above. Since T will not in general be complete with respect to $\mathfrak{a}$ the above result is not too surprising. One can still ask if the incompleteness of (W,T) arises entirely from the incompleteness of T or whether the axioms and rules of inference of W are partially responsible? Two approaches to this question have been proposed in the literature. The first is the notion of an adequate proof rule and is due to J.W. deBakker. The second solution is a type of relative completeness theorem first proposed by Stephen Cook. We examine deBakker's idea first.

3.5.1 Definition:

Suppose that L_A is expressive relative to L_E and $\mathfrak{a}$. Let

$$\frac{H_1,\ldots,H_n}{\{P\}\ A\ \{Q\}}\ (*)$$

be a simple rule of inference for the G.C.S. A. Let $U_0 = P, U_1, \ldots U_M = Q$ be the predicate symbols appearing in the H_i but not in A itself. Then $(*)$ is a fully adequate proof rule for A iff

(A) each program statement occurring in some Hypothesis H_i occurs syntactically as a proper substatement of A[1].

(B) whenever $\{\hat{P}\}\ A\ \{\hat{Q}\}$ is true, there exist predicates $\hat{U}_0,\ldots\hat{U}_M$ expressible by formulas of L_E such that

(i) $\hat{P} \rightarrow \hat{U}_0$

(ii) $\hat{U}_M \rightarrow \hat{Q}$

(iii) all of the Hypothesis $H_1,H_2,\ldots,H_N$ will be true if $\hat{U}_0,\ldots\hat{U}_M$ are substituted for $U_0,\ldots,U_M$.

Cook gets around the problem mentioned above by assuming the existance of a complete proof system T for L_A, or equivalently an oracle for deciding the truth (relative to $\mathfrak{A}$) of closed formulas in L_A. The following simple theorem serves to connect these two approaches.

3.5.2 Theorem:

Consider the programming language $PL[L_E,L_A: C_1,\ldots,C_k]$. Suppose that L_A is expressive relative to L_E and $\mathfrak{A}$. Let T be a complete proof system for L_A and $\mathfrak{A}$. Let $R_1,\ldots,R_k$ be simple rules of inference for the GCS's $C_1,\ldots,C_k$ which are both sound and fully adequate. Let A be a statement of a $PL[L_E,L_A: C_1,\ldots,C_k]$ program in which no procedure declarations occur (other than those implicitly occuring because of the GCS's $C_1,\ldots,C_k$). Then $\{P\}\ A\ \{Q\}$ is true with respect to $\mathfrak{A}$ iff $\vdash_{(W,T)} \{P\}\ A\ \{Q\}$ where P and Q are formulas of L_A and W consists of axioms and rules of the Basic set plus the rules $R_1,\ldots R_k$.

---

[1] This condition is needed to rule out trivial rules of inference such as $\frac{\{P\}\ A\ \{Q\}}{\{P\}\ A\ \{Q\}}$ which otherwise would satisfy condition B.

Proof: The "only if" part of this theorem was discussed earlier in this chapter when we introduced the notions of soundness. The "if" part of the theorem is proved by structural induction on A.

Case(1): A is the assignment statement u := t. {P} u :=t {Q} is true, so $P \rightarrow R[u := t](Q)$. $R[u := t](Q) = Q\frac{t}{u}$. Thus $P \rightarrow Q\frac{t}{u}$ is a true formula of L_A and $\{Q\frac{t}{u}\}$ u := t {Q} is an instance of the axiom for the assignment statement. By the rule of consequence {P} u := t {Q} is provable.

Case (2): A is a conditional statement of the form $(b \rightarrow A_1, A_2)$. Since $\{P\}\ (b \rightarrow A_1, A_2)\ \{Q\}$ is true we have $\{P\} \rightarrow RA(Q)$ or $P \rightarrow [b \wedge RA_1(Q)] \vee [\sim b \wedge RA_2(Q)]$.

Thus $P \wedge b \rightarrow RA_1(Q)$ and $P \wedge \sim b \rightarrow RA_2(Q)$ are both true. By the inductive assumption $\vdash\{P \wedge b\}\ A_1\ \{Q\}$ and $\vdash\{P \wedge \sim b\}\ A_2\ \{Q\}$. By rule of inference (3) we get that $\vdash\{P\}\ (b \rightarrow A_1, A_2)\ \{Q\}$ or $\vdash\{P\}\ A\ \{Q\}$.

Case (3): A is a composite statement $A = (A_1; A_2)$. Since {P} A {Q} is true, we have that $\{P\}\ A_1\ \{RA_2(Q))\}$ and $\{RA_2(Q)\}\ A_2\ \{Q\}$ are both true and $RA_2(Q)$ is expressible by a formula of L_A. By the inductive assumption $\vdash\{P\}\ A_1\ \{RA_2(Q)\}$ and $\vdash\{RA_2(Q)\}\ A_2\ \{Q\}$. Hence by rule (2) for the composition of statements $\vdash\{P\}\ (A_1; A_2)\{Q\}$ or $\vdash\{P\}\ A\ \{Q\}$.

Case (4): Suppose $A = C_i[b_i, \ldots, b_m; A_1 \ldots A_n]$ and {P} A {Q} is true. Then by full adequacy of R_i there exist formulas $U_0, \ldots U_\ell$ in L_A such that

(a) $P \rightarrow U_0$ is true in L_A (w.r.t. $\mathfrak{A}$)

(b) $U_\ell \rightarrow Q$ is true in L_A (w.r.t $\mathfrak{A}$)

(c) $H_0, \ldots, H_{n_i}$ (the hyptothesis of R_i) are all true (w.r.t. $\mathfrak{A}$).

Since R_i is a simple rule of inference $H_0, \ldots, H_{n_i}$; all are formulas of L_A or have the form $\{V_1\}\ B\ \{V_2\}$ where V_1, V_2 are formulas of L_A and B is a statement with simpler structure than A. By the inductive

assumption $\vdash H_j$, $0 \leq j \leq n_i$. By rule R_i we get that $\vdash \{P\}\ A\ \{Q\}$.

Since A cannot be a procedure call, cases (1) - (4) include all possibilities and the proof is complete.

To illustrate how one proves that a rule of inference is adequate, we return to the example at the end of section 3.3. There we considered the GCS with prototype:

$$C_2 = \langle \{X \equiv (b_1 \rightarrow A_1;\ Y, A_2),\ \ Y \equiv (b_2 \rightarrow A_2;\ X, Y)\},\ X \rangle$$

We showed that the rule of inference

$$\frac{\{P \wedge b_1\} A_1 \{U\},\ \{P \wedge \sim b_1\}\ A_2\ \{Q\},\ \{U \wedge b_2\}\ A_2\ \{P\}}{\{P\}\ C_2\ [b_1, b_2;\ A_1, A_2]\ \{Q\}}$$

for C_2 was sound. We now show that this rule of inference is fully adequate. Assume that $\{P\}\ C_2\ [b_1, b_2;\ A_1, A_2]\ \{Q\}$ is true. Then $P \rightarrow RX(Q)$ is true in $\mathfrak{a}$. By the first predicate transformer fixed point theroem we have the $\langle RX(Q),\ RY(Q) \rangle$ is a fixed point of the system:

$$G_1(V,W) = [b_1 \wedge RA_1(W)] \vee [\sim b_1 \wedge RA_2(Q)]$$

$$G_2(V,W) = [b_2 \wedge RA_2(V)] \vee [\sim b_2 \wedge W]$$

Furthermore, note that if L_A is expressive relative to L_E and $\mathfrak{a}$, then $RX(Q)$ and $RY(Q)$ will be expressible by formulas of L_A. Thus

(a) $P \rightarrow RX(Q)$

(b) $Q \rightarrow Q$

and (c) (i) $RX(Q) \wedge b_1 \rightarrow RA_1(RY(Q))$

(i.e. $\{RX(Q) \wedge b_1\}\ A_1\ \{RY(Q)\}$)

(ii) $RX(Q) \wedge \sim b_1 \rightarrow RA_2(Q)$

(i.e. $\{RX(Q) \wedge \sim b_1\}\ A_2\ \{Q\}$)

(iii) $RY(Q) \wedge b_2 \rightarrow RA_2(RX(Q))$

(i.e. $\{RY(Q) \wedge b_2\}\ A_2\ \{RX(Q)\}$)

3.6. Generating Hoare-like Rules of Inference

In this section we give a simple algorithm for generating Hoare-like rules of inference for GCS's which have as their declaration part a regular system of procedure declarations. The rules of inference generated in this manner will be shown to be both sound and complete. Because of the relationship between regular systems of procedure declarations and flowchart schemes, this section can in some sense be viewed as a restatement of the inductive assertion method [FL67] and a proof of its soundness and completeness. We believe that the present method of proof demonstrates the usefulness of the predicate transformer fixed point theorem discussed in Chapter 2.

The algorithm is stated in a form which simplifies the proof of soundness and full adequacy so that some of the steps are redundant and could be eliminated or simplified if desired. A similar remark applies to the rules of inference generated by the algorithm--in many cases several assertions generated by the algorithm could be combined into a single assertion to obtain a more concise rule of inference.

Let $C = \langle H; X_1; \beta_1, \ldots, \beta_M; \alpha_1, \ldots, \alpha_N \rangle$ be the GCS under consideration. Thus H consists of a set of procedure declarations:

$$X_1 \equiv \tau_1$$
$$\vdots$$
$$X_K \equiv \tau_K$$

which is both self-contained and regular. The algorithm uses a set TOP which ultimately will contain the required hypothesis of the rule of inference.

Initialization:

Let TOP contain the formula $P \to U_1$ of L_A together with the triples

$[U_1,\tau_1,Q]$

$[U_2,\tau_2,Q]$

$\vdots$

$[U_K,\tau_K,Q]$

(i.e. one for each procedure declaration; $U_1,\ldots,U_K$ are new predicate symbols)

Algorithm

(A) Apply transformation 1-5 below to the elements of the set TOP until no further such transformations are applicable (the order in which these transformations are made will turn out to be unimportant).

(B) When no transformations are applicable, the elements of TOP are the hypothesis of the desired rule for C, i.e. if TCP = $\{t_1,\ldots,t_s\}$ then the rule for C is

$$\frac{t_1,\ldots,t_s}{\{P\}\ C\ [\beta_1,\ldots;\ \alpha_1,\ldots]\{Q\}}$$

Transformations on the set TOP:

If a transformation has the form

$$t \Longrightarrow t_1,\ldots,t_k$$

then an occurence of $t \in$ TOP may be replaced by $t_1,\ldots,t_k$.

(1) $[U,I,V] \Rightarrow U. \to V$

(2) $[U, (S_1;S_2), V] \Rightarrow [U,S_1,W], [W, S_2, V]$

where W is a new predicate symbol

(3) $[U, (b \to S_1,S_2), V] \Rightarrow [U \wedge b,S_1,V], [U \wedge \sim b, S_2, V]$

(4) $[U,X_i,V] \Rightarrow U \to U_i$ where X_i is the name of a procedure whose

declaration occurs in H.

(5) $[U,A,V] \Rightarrow \{U\}\ A\ \{V\}$ where A is an atomic statement other than a procedure call.

Note that when none of the above transformations are applicable, the elements of TOP will contain no occurences of procedure names.

Note also that since we are dealing with a regular GCS, the algorithm will never encounter an element of TOP of the form $[U,X_i,V]$ where X_i is a procedure name and V is different from Q.

An an example consider the GCS

$$C_1 = \langle \{X = (\beta \rightarrow \alpha\ ;\ X,\ I)\},\ X,\beta,\alpha \rangle$$

TOP	Transformation No.
$P \rightarrow U,\ [U,\ (\beta \rightarrow \alpha;\ X,\ I)\ Q]$	initialization
$P \rightarrow U,\ [U \wedge \sim \beta,\ I,\ Q],\ [U \wedge \beta,\ (\alpha;\ X),\ Q]$	3
$P \rightarrow U,\ U \wedge \sim \beta \rightarrow Q,\ [U \wedge \beta,\ (a;X),\ Q]$	I
$P \rightarrow U,\ U \wedge \sim \beta \rightarrow Q,\ [U \wedge \beta,\ \alpha,\ V],\ [V,\ X,\ Q]$	2
$P \rightarrow U,\ U \wedge \sim \beta \rightarrow Q,\ [U \wedge \beta,\ \alpha\ ,\ V],\ V \rightarrow U$	4
$P \rightarrow U,\ U \wedge \sim \beta \rightarrow Q,\ \{U \wedge \beta\}\ \alpha\ \{V\},\ V \rightarrow U$	5

Thus the rule of inference for C_3 should be

$$\frac{P \rightarrow U,\ \ U \wedge \sim \beta \rightarrow Q,\ \ V \rightarrow U,\ \ \{U \wedge \beta\}\ \alpha\ \ \{V\}}{\{P\}\ C_1\ [\alpha;\beta]\ \ \{Q\}}$$

which may be simplified to obtain the usual axiom for the while statement

$$\frac{P \to U,\ U \wedge \sim \beta \to Q,\ \{U \wedge \beta\}\ \alpha\ \{U\}}{\{P\}\ C_1\ [\alpha;\beta]\ \{Q\}}$$

We leave it to the reader to try examples involving more than one procedure declaration.

We now show that the above algorithm generates axioms which are both sound and fully adequate. The proof uses a version of the predicate transformer fixed point theorem which we restate for convenience below:

Let $\begin{cases} X_1 \equiv \tau_1 \\ \vdots \\ X_K \equiv \tau_K \end{cases}$ be a regular system of procedure declarations.

and let $Q \subseteq S$ be a predicate. We associate with * and Q a set of K predicate transformers $G_1, G_2 \ldots G_K$ where each $G_i : P(S) \to P(S)$, the G_i's are defined in terms of a mapping $\Gamma : ST \times P(S) \to P(S)$, i.e. for all i $G_i(U_1, U_2, \ldots U_K) = \Gamma[\tau_i](Q)$ where Γ is defined by cases

(a) $\Gamma[A_1, A_2)]\ (W) = \Gamma[A_1]\ (\Gamma\ [A_2](W))$

(b) $\Gamma[(b \to A_1, A_2)](W) = (b \wedge \Gamma[A_1](W)) \vee (\sim b \wedge \Gamma[A_2]\ (W))$

(c) $\Gamma[A](W) = R[A](Q)$, A atomic

(d) $\Gamma[X_i](W) = U_i$

Theorem:

Let (*), Q, $G_1, \ldots, G$ be as described above.
Define $G: P(S)^K \to P(S)^K$ by $G(U_1, \ldots, U_K) = \langle G_1(U_1, \ldots, U_K), \ldots, G_K(U_1, \ldots, U_K)\rangle$. Then $\langle RX_1(Q), \ldots, RX_K(Q)\rangle$ is the greatest fixed point of G under the natural ordering on $P(S)^n$.

We consider the proof of soundness first. In order to make the proof clearer, we apply the following notational transformation to the algorithm: replace $[U,\tau,V]$ by $U \to \Gamma[\tau](V)$ where Γ is the mapping associated with predicate transformers G_i in the statement of the theorem

above. We will show that when the algorithm terminates $P \rightarrow R\,[X_1](Q)$ will be a logical consequence of the assertions in the set TOP. More precisely let TOP_r be the elements of the set TOP after r transformations have been applied, then we will show that:

$$P \rightarrow U_1,$$
$$U_1 \rightarrow G_1(U_1,\ldots,U_K),$$
$$\vdots$$
$$U_K \rightarrow G_K(U_1,\ldots,U_K)$$

must all be true if the assertions in TOP are true. By the maximal fixed point characterization of $RX_i(Q)$ it follows that $U_1 \rightarrow RX_1(Q)$ is true and therefore $P \rightarrow RX_1(Q)$ also.

First observe that this condition holds initially. $P \rightarrow U_1$ is in TOP_o. Since $U_i \rightarrow \Gamma[\tau_i](Q)$ is in TOP_o for $1 \leq i \leq K$, we have that $U_i \rightarrow G_i(U_1,\ldots,U_K)$ is true for $1 \leq i \leq K$ (by definition $G_i(U_i,\ldots,U_K) = \Gamma\,[\tau_i](Q)$). Next assume that the condition above is true of TOP_{n-1}, we show that it will also be true of TOP_n. We proceed by examining the ways in which TOP_n may be obtained from TOP_{n-1}.

(1) $U \rightarrow \Gamma[I](V) \Rightarrow U \rightarrow V$: If $U \rightarrow V$ is true, then $U \rightarrow \Gamma[I](V)$ will be true also since $\Gamma[I](V) = R[I](V) = V$.

(2) $U \rightarrow \Gamma[(S_1;\ S_2)](V) \Rightarrow U \rightarrow \Gamma[S_1](W),\ W \rightarrow \Gamma[S_2](V)$:
It is easily checked that $\Gamma[A](.)$ is monotonic, thus if $U \rightarrow \Gamma[S_1](W)$ and $W \rightarrow \Gamma[S_2](V)$ are true, it follows that $U \rightarrow \Gamma[S_1](\Gamma\,[S_2](V))$ or $U \rightarrow \Gamma[(S_1;\ S_2)](V)$ is true also.

(3) $U \rightarrow \Gamma[(b \rightarrow S_1,\ S_2)](V) \Rightarrow U \wedge b \rightarrow \Gamma[S_1](V),\ U \wedge \sim b \rightarrow \Gamma[S_2](V)$:
if $U \wedge b \rightarrow \Gamma\,[S_1](V)$ and $U \wedge \sim b \rightarrow \Gamma[S_2](V)$ are true then $U \rightarrow (b \wedge \Gamma[S_1](V)) \vee (\sim b \wedge \Gamma[S_2](V))$ or $U \rightarrow \Gamma[(b \rightarrow S_1, S_2)](V)$ is true also.

(4) $U \rightarrow \Gamma[X_i](V) \Longrightarrow U \rightarrow U_i$: Since H is a regular system of procedure declarations we must have $V = Q$. Since $\Gamma[X_i](Q) = U_i$ it follows that if $U \rightarrow U_i$ is true, $U \rightarrow \Gamma[X_i](V)$ will be true also.

(5) $U \rightarrow \Gamma[A](V) \Longrightarrow \{U\}\ A\ \{V\}$: If $\{U\}\ A\ \{V\}$ is true then $U \rightarrow RA(V)$ holds. Since A is atomic and $\Gamma[A](V) = R[A](V)$, the desired result follows.

This completes the proof of soundness. We next show that the rules of inference generated by the algorithm are <u>fully adequate</u>. Let $\hat{P}$ and $\hat{Q}$ be predicates such that $\hat{P} \rightarrow R[C[b_1,\ldots,; A_1,\ldots]]\ (\hat{Q})$ (i.e. $\hat{P} \rightarrow R[X_i](\hat{Q})$) is true and $\hat{P}$ and $\hat{Q}$ are expressible by formulas of L_A. Assume that at the rth stage in the execution of the algorithm the predicate symbols $P = V_0,\ldots,V_k = Q$ occur in the assertions of TOP_r but not in $C[b_1,\ldots A_1,\ldots,]$. We show that there are predicates $\hat{P} = \hat{V}_0,\ldots,\hat{V}_k = \hat{Q}$ which are expressible by formulas of L_A and have the property that all of the assertions in TOP_r will be satisfied if $\hat{V}_0,\ldots,\hat{V}_k$ are substituted for $V_0,\ldots,V_k$.

Note that this condition holds initially. Choose $\hat{U}_1 = RX_1(\hat{Q}),\ldots,$ $\hat{U}_K = RX_K(\hat{Q})$ then $\hat{P} \rightarrow RX_1(\hat{Q})$ is true and since $G_i(RX_i(\hat{Q}),\ldots,$ $RX_K(\hat{Q})) = RX_i(\hat{Q})$ we also have that $\hat{U}_i \rightarrow \Gamma[\tau_i](\hat{Q})$ is true. As in the proof of soundness we assume that the condition holds for TOP_{n-1} and show that it holds for TOP_n as well by checking each of the possible cases by which TOP_n could have been obtained from TOP_{n-1}. Thus for example in case 2: $U \rightarrow \Gamma[(S_1,S_2)](V) \Rightarrow U \rightarrow \Gamma[S_1](W),\ W \rightarrow \Gamma[S_2](V)$-- merely choose $\hat{W} = \Gamma[S_2](V)$ and observe that all of the required conditions will be satisfied of TOP_n if they were satisfied by TOP_{n-1}. The other cases are very similar.

CHAPTER 4

NON-REGULAR CONTROL STRUCTURES

4.1 Introduction

In this chapter we show that more powerful axioms are needed in order tc obtain sound, complete axiom systems for non-regular control structures. Using the tools of Chapter 2 we examine methods of obtaining such axioms which have been proposed by deBakker and Meertens [DE73] and by Gorelick [GO75]. In particular we examine the claim of deBakker and Meertens that an infinite pattern of precondition-postcondition assertions is needed when extending the inductive assertion method to handle non-regular control structures. We show that the argument for the necessity of such assertion patterns is based on a treatment of predicates as sets of program states and of programs as state transformations. If predicates are throught of as formulas in first order predicate calculus, then substantially simpler rules of inference may be constructed which are both sound and complete.

4.2 Non-Regular Systems and the Necessity of More Complicated Rules of Inference

If we apply the algorithm of the preceding section to the (non-regular) precedure definition

$$X \equiv b \to A_1\ X\ A_2,\ I$$

we obtain the rule of inference

(i) $$\frac{P \to U,\ \{U \wedge b\}\ A_1\{W_1\},\ \ W_1 \to U,\ \ Q \to W_2,\ \{W_2\}\ A_2\ \{Q\}}{\{P\}\ X\ \{Q\}}$$

It is not difficult to show that (i) is sound--argument almost identical

to the one used in section 3.3 works. Unfortunately it is also easy to show that (i) cannot be fully adequate. To see that this is the case we first simplify (i) to obtain

(ii) $$\frac{P \to U,\ \{U \wedge b\}\ A_1\ \{U\},\ \{Q\}\ A_2\ \{Q\},\quad U \wedge \sim b \to Q}{\{P\}\ X\ \{Q\}}$$

(ii) is also sound and will be fully adequate iff (i) is.

Next consider the procedure definition $Y \equiv b \to A_1\ Y,\ A_2$. Note that Y has a regular definition and the the algorithm of Chapter 3 is known to apply in this case. Thus we obtain the rule of inference:

$$\frac{P \to U,\ \{U \wedge b\}\ A_1\ \{W\}\ ,\quad W \to U,\quad Q \to Q,\ \{U \wedge \sim b\}\ A_2\ \{Q\}}{\{P\}\ Y\ \{Q\}}$$

which is known to be both sound and fully adequate.

We simplify to obtain

(iii) $$\frac{P \to U,\ \{U \wedge b\}\ A_1\ \{U\},\ \{U \wedge \sim b\}\ A_2\ \{Q\}}{\{P\}\ Y\ \{Q\}}$$

Let P_0, Q_0 and $\mathfrak{a}$ be such that $\{P_0\}\ X\ \{Q_0\}$ is true but $\{P_0\}\ Y\ \{Q_0\}$ is false. For example, suppose that

$b \equiv n > 0$

$A_1 \equiv n := n-1$

$A_2 \equiv n := n+1$

$P_0 \equiv \{n = n_0\}$

$Q_0 \equiv \{n = n_0\}$

If (ii) is fully adequate, there would be a formula U_0 expressible in the assertion language L_A such that

$$P_0 \to U_0,\ \{U_0 \wedge b\}\ A_1\ \{U_0\},\ \{Q_0\}\ A_2\ \{Q_0\},\quad U_0 \wedge \sim b \to Q_0$$

are all true. But then we would also have

$$P_0 \to U_0,\ \{U_0 \wedge b\}\ A_1\ \{U_0\},\ \{U_0 \wedge \sim b\}\ A_2\ \{Q_0\}$$

Thus, by (iii) we would be able to conclude $\{P_0\}\ Y\ \{Q_0\}$ but this is a contradiction.

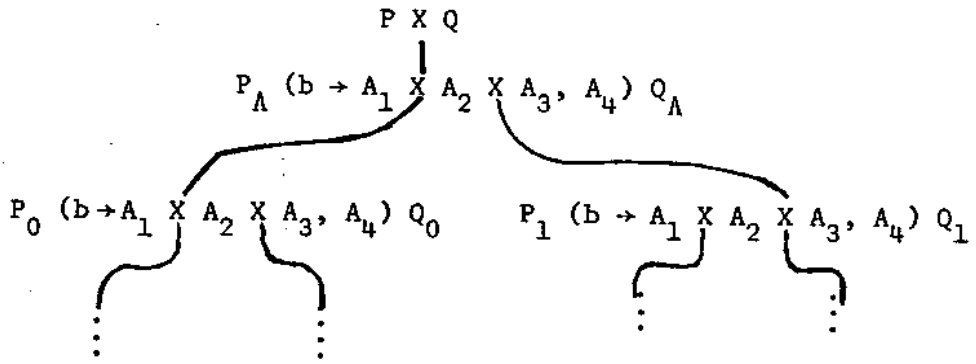
A more general version of the above argument is used by Fokkinger [FO73] and by deBakker and Meertens [DE73] to argue that an <u>infinite pattern of assertions</u> is <u>necessary</u> in order to obtain the type of completeness that we desire. In their paper on the "Completeness of the Inductive Assertion Method", deBakker and Meertens suggest an infinite collection of assertions determined by attaching a pair of assertions to each node in the infinite tree obtained by unwinding the recursive definition ("the tree of incarnations"). Thus in the case of $X = b \to A_1\ X\ A_2$, I the tree of incarnations has the form

$$\begin{array}{l} P\ \ X\ \ q \\ \quad\ \ | \\ P_0\ (b \to A_1\ X\ A_2,\ I)\ q_0 \\ \quad\ \ | \\ P_1\ (b \to A_1\ X\ A_2,\ I)\ q_1 \\ \quad\ \ | \\ P_1\ (b \to A_1\ X\ A_2,\ I)\ q_2 \\ \quad\ \ \vdots \end{array}$$

so that we could expect to obtain the rule of inference

$$\left.\begin{array}{l} P \to P_0, \\ \{P_i \wedge b\}\ A_1\ \{P_{i+1}\}, \\ \{Q_{i+1}\}\ A_2\ \{Q_i\}, \\ P_i \wedge \sim b \to Q_i, \\ Q_0 \to Q \end{array}\right\} \quad i \in N$$

For $X = b \to A_1\ X\ A_2\ X\ A_3, A_4$ the tree of incarnations is more complicated:



The assertions P_σ, Q_σ may indexed by strings σ in $(0+1)^*$ and we obtain the rule of inference

$P \to P_\Lambda$,

$\{P_\sigma \wedge b\}\ A_1\ \{P_{\sigma 0}\}$,

$\{P_\sigma \wedge \sim b\}\ A_4\ \{Q_\sigma\}$

$\{Q_{\sigma 0}\}\ A_2\ \{P_{\sigma 1}\}$ $\qquad \sigma \in (0+1)^*$

$\{Q_{\sigma 1}\}\ A_3\ \{Q_\sigma\}$,

$Q_\Lambda \to Q$

$\{P\}\ X\ \{Q\}$

To illustrate the methods of [DE73] we prove (using the notation of this paper) that the rule of inference given above for $X = b \to A_1\ X\ A_2\ X\ A_3, A_4$ is both sound and fully adequate.

First for soundness, we assume that

(a) $P \to P_\Lambda$

(b) $P_\sigma \wedge b \to R\ [A_1]\ (P_{\sigma 0})$

(c) $P_\sigma \wedge \sim b \to R[A_1]\ (Q_\sigma)$

(d) $Q_{\sigma 0} \to R[A_2]\ (P_{\sigma 1})$

(e) $Q_{\sigma 1} \to R\ [A_3]\ (Q_\sigma)$

(f) $Q_\Lambda \to Q$

are true and show that $P \to RX(Q)$ must be true also. The argument is based on the fundamental invariance theorem. Assume that for all $\sigma \in (0+1)^*$ we have $P_\sigma \to RX_i(Q_\sigma)$ then

$P_{\sigma 0} \to RX_i(Q_{\sigma 0})$

$P_{\sigma 0} \to RX_i\ (RA_2(P_{\sigma 1}))$ (by d)

$RA_1\ (P_{\sigma 0}) \to RA_1\ RX_i\ RA_2\ (P_{\sigma 1})$ (by monotonicity)

(*) $P_\sigma \wedge b \to RA_1\ RX_i\ RA_2\ (P_{\sigma 1})$ (by b)

Similarly, $P_{\sigma 1} \to RX_i(Q_{\sigma 1})$

So $P_{\sigma 1} \to RX_i\ (RA_3(Q_\sigma))$ (by e)

Combining this with (*) we get

$$P_\sigma \wedge b \to RA_1\ RX_i\ RA_2\ (RX_i\ RA_3\ (Q_\sigma))$$

$$P_\sigma \wedge \sim b \to RA_4\ (Q_\sigma) \quad \text{(by c)}$$

Thus, $P_\sigma \to [b \wedge RA_1\ RX_i\ RA_2\ RX_i\ RA_3\ (Q)] \vee [\sim b \wedge RA_4\ (Q_\sigma)]$

or $P_\sigma \to RX_{i+1}(Q_\sigma)$ •

So $P_\sigma \to RX(Q_\sigma)$ for all $\sigma \in (0+1)^*$; taking $\sigma = \Lambda$ and making use of (a) and (f) we get $P \to RX(Q)$ as required.

We turn next to <u>full adequacy</u>. Note that

$$RX(Q) = [b \wedge RA_1\ RX\ RA_2\ RX\ RA_3\ (Q)] \vee [\sim b \wedge RA_4\ (Q)]$$

Let the sequence $Q_{\sigma 1}$, $\sigma \in (0+1)^*$ be defined by

$$Q_\Lambda = Q$$

$$Q_{\sigma 0} = RA_2\ RX\ RA_3\ (Q_\sigma)$$

$$Q_{\sigma 1} = RA_3\ (Q_\sigma)$$

Let the sequence $P_{\sigma 1}$, $\sigma \in (0+1)^*$ be defined in terms of the sequence Q by

$$P_\sigma = RX\ (Q_\sigma)$$

Note that if L_A is expressive relative to L_E and $\mathfrak{a}$, then P_σ, Q_σ $\sigma \in (0+1)^*$ will be expressible by formulas of L_A. Then

(A) $P \rightarrow RX(Q) = P_\Lambda$

(B) $P_\sigma \wedge b = RX(Q_\sigma) \wedge b$

$= RA_1 (RX (RA_2 (RX (RA_3 (Q_\sigma)))))$

$= RA_1 (RX (Q_{\sigma 0}))$

$= RA_1 (P_{\sigma 0})$

(C) $Q_{\sigma 0} = RA_2 (RX (RA_3 (Q_\sigma)))$

$= RA_2 (RX (Q_{\sigma 1}))$

$= RA_2 (P_{\sigma 1})$

(D) $Q_{\sigma 1} = RA_3 (Q_\sigma)$

(E) $P_\sigma \wedge \sim b \rightarrow RA_4 (Q_\sigma)$

(F) $Q_\Lambda \rightarrow Q$ since $Q_\Lambda = Q$

Thus, all of the hypothesis of the rule of inference for $X = b \rightarrow A_1 X A_2 X A_3, A_4$ are satisfied and the proof of full adequacy is complete.

The argument used be deBakker and Meertens for the necessity of rules of inference such as those discussed above is based on the notion that predicates are merely sets of states. If on the other hand predicates are thought of as formulas in the first order predicate calculus and the role of non-active variables is taken into account then substantially simpler rules of inference can be constructed. Alternatively these rules of inference may be viewed as ways of organizing or indexing the predicates used by deBakker and Meertens.

4.3 Axioms for Non-Regular GCS's

In order to prove soundness and completeness of axioms for non-regular GCS's additional machinery is needed. In this section we introduce this additional machinery and apply it to a simple example of a non-regular GCS. This example will serve as a prototype for our considerations of the general case in the next section. We begin with a number of definitions

4.3.1 Definition:

Let A be a statement in a PL program E. A variable u is active in A iff either

(i) A is an assignment statement and u occurs in the right or left side of A.

(ii) A is a conditional statement $b \rightarrow A_1, A_2$ and either u occurs in b or u is active in A_1 or u is active in A_2.

(iii) A is a composite statement $(A_1; A_2)$ and u is active in A_1 or u is active in A_2.

(iv) A is a call on the procedure X with declaration $X \equiv \tau$ occuring in the declaration part of E and u is active in τ.

If u is not active in A then u is said to be non-active. A term is non-active if the only variables occurring in it are non-active variables. Finally, a substitution σ is an admissible substitution if σ is a substitution of non-active terms for non-active variables.

Thus, for example, if $E = \langle H, A \rangle$ is the program of Section 2.2 then u_0, v_0, and w_0 are non-active variables, w_0 and $g(f(u_0, v_0))$ are non-active terms and $\sigma = \frac{w_0,\ h(u_0, g(v_0))}{u_0, v_0}$ is an admissible substitution.

4.3.2 Proposition:

Let A be a statement in a PL program E and let P be a formula of L_A which does not contain any free variables that are active in A. Then

$$F\,[A]\,(P \wedge Q) = P \wedge F\,[A]\,(Q)$$

Proof: $F\,[A]\,(P \wedge Q) \subseteq F\,[A]\,(P) \wedge F\,[A]\,(Q)$

By the restriction on P in the hypothesis of the theorem $\{P\}\ A\ \{P\}$ is true and hence $F\,[A]\,(P) \subseteq P$. Thus $F\,[A]\,(P \wedge Q) \subseteq P \wedge F\,[A]\,(Q)$.

Conversely, let $s \in P \wedge F[A](Q)$ then $s \in P$ and $s \in F[A](Q)$. Thus there is an s' such that $s = M[A](s')$ and $s' \in Q$. By the restriciton on P, s and s' can only differ on variables not occurring free in P. Thus $s' \in P$ also. Therefore $s' \in P \wedge Q$ and $s \in F[A](P \wedge Q)$. It follows that $P \wedge F[A](Q) \subseteq F[A](P \wedge Q)$.

4.3.3 Proposition

Let A be a statement in a PL program E. Let σ be an admissible substitution with respect to A. Let Q be a formula of the assertion language L_A. Then $F[A](Q)\sigma = F[A](Q\sigma)$.

Proof: Let $\bar{u}$ contain the variables which are active in A. Let $\bar{u}_0$ contain new variables (i.e. not appearing in E, σ, or Q) and have the same size as $\bar{u}$. Then

$$F[A](Q(\bar{u}))\sigma = F[A](\exists \bar{u}_0 [\bar{u} = \bar{u}_0 \wedge Q(\bar{u}_0)])\sigma$$
$$= \exists \bar{u}_0 [F[A](\bar{u} = \bar{u}_0 \wedge Q(\bar{u}_0))]\sigma$$
$$= \exists \bar{u}_0 [F[A](\bar{u} = \bar{u}_0) \wedge Q(\bar{u}_0)]\sigma$$
$$= \exists \bar{u}_0 [F[A](\bar{u} = \bar{u}_0)_\sigma \wedge Q(\bar{u}_0)\sigma]$$

But $F[A](\bar{u} = \bar{u}_0)\sigma = F[A](\bar{u} = \bar{u}_0)$ and $Q(\bar{u}_0)\sigma$ does not contain any free variables which are active in A since σ is admissible.

Thus, $F[A](Q(\bar{u}))\sigma = \exists \bar{u}_0 [F[A](\bar{u} = \bar{u}_0) \wedge Q(\bar{u}_0)\sigma]$
$$= \exists \bar{u}_0 [F[A](\bar{u} = \bar{u}_0 \wedge Q(\bar{u}_0)\sigma)]$$
$$= F[A](\exists \bar{u}_0 [\bar{u} = \bar{u}_0 \wedge Q(\bar{u}_0)\sigma])$$
$$= F[A](Q(\bar{u})\sigma)$$

To illustrate the problems which occur when dealing with non-regular GCS's we consider the GCS C_3 with prototype

$$C_3 = \langle \{X = b \rightarrow A_1; X; A_2, I\}; X \rangle$$

We will show that a sound and complete rule of inference for C_3 is

$$(*)\ \frac{\{P \wedge b\}\ A_1\ \{\exists \bar{u}_0\ [P\ \frac{\bar{u}_0}{\bar{v}_0} \wedge T]\},\{\exists u_0\ [Q\ \frac{\bar{u}_0}{\bar{v}_0} \wedge T]\}\ A_2\ \{Q\},\ P \wedge \sim b \rightarrow Q}{\{P\}\ C_3\ [b:\ A_1;\ A_2]\ \{Q\}}$$

where $\bar{v}_0$ are the variables which occur free in the predicates P and Q but are not active in $C_3\ [b:\ A_1;\ A_2]$. $\bar{u}_0$ consists entirely of new variables. T is a predicate which does not contain any active free variables.[1]

In order to prove that (*) is sound we assume

(i) $FA_1\ [P \wedge b] \rightarrow \exists \bar{u}_0\ [P\ \frac{\bar{u}_0}{\bar{v}_0} \wedge T]$

(ii) $FA_2\ (\exists \bar{u}_0\ [Q\ \frac{\bar{u}_0}{\bar{v}_0} \wedge T]\ ) \rightarrow Q$

and

(iii) $P \wedge \sim b \rightarrow Q$

and deduce $FX(P) \rightarrow Q$ by means of the fundamental invariance theorem.

Define the sequence $\{X^i\}_{i \geq 0}$ by

$$X^0 = \Omega$$

$$X^{i+1} = b \rightarrow A_1;\ X^i;\ A_2,\ I$$

So that $FX^{i+1}(P) = FA_2\ (FX^i\ (FA_1\ (P \wedge b))) \vee (P \wedge \sim b)$.

If $FX^i(P) \rightarrow Q$

then $\exists \bar{u}_0\ [FX^i(P)\ \frac{\bar{u}_0}{\bar{v}_0} \wedge T] \rightarrow \exists \bar{u}_0\ [Q\ \frac{\bar{u}_0}{\bar{v}_0} \wedge T]$

so, by propositions 4.3.2, 4.3.2 and 4.3.11.

$$FX^i\ (\exists \bar{u}_0\ [P\ \frac{\bar{u}_0}{\bar{v}_0} \wedge T]) \rightarrow \exists \bar{u}_0\ [Q\ \frac{\bar{u}_0}{\bar{v}_0} \wedge T]$$

[1] Intuitively, the predicate T enables information to be transferred across a procedure call. If the effect of A_1 on the program variables can be described by a simple admissable substitution σ, then T may be eliminated and a simpler axiom involving the substitution σ will work.

By (i) and monotonicity of $FX^i(\cdot)$

$$FX^i\ (FA_1(P \wedge b)) \rightarrow \exists \bar{u}_0\ [Q\ \frac{\bar{u}_0}{\bar{v}_0} \wedge T]$$

By monotonicity of $FA_2(\cdot)$

$$FA_2(FX^i\ (FA_1(P \wedge b))) \rightarrow FA_2\ (\exists \bar{u}_0\ [Q\ \frac{\bar{u}_0}{\bar{v}_0} \wedge T]\ )$$

By (ii) and (iii) we obtain

$$FA_2(FX^i\ (FA_1\ (P \wedge b))) \vee (P \wedge \sim b) \rightarrow Q$$

So $FX^{i+1}(P) \rightarrow Q$. Hence by the fundamental invariance theorem we conclude that $FX(P) \rightarrow Q$ holds.

To prove completeness we need several new axioms in addition to those of the basic set given in section 3.3 The new axioms are:

R1. $\{T\}\ A\ \{T\}$ provided that T does not contain any free variables which are active in A.

R2. $$\frac{\{P_1\}\ A\ \{Q_1\},\ \{P_2\}\ A\ \{Q_2\}}{\{P_1 \wedge P_2\}\ A\ \{Q_1 \wedge Q_2\}}$$

R3. $$\frac{\{P(\bar{u}_0)\}\ A\ \{Q(\bar{u}_0)\}}{\{\exists \bar{u}_0\ P(\bar{u}_0)\}\ A\ \{\exists \bar{u}_0\ Q(\bar{u}_0)\}}$$

provided that the variable u_0 is not active in A[1].

---

[1] This axiom is not absolutely essential. However its use simplifies the statements of some of the other axioms and also the proof of the completeness theorem.

Suppose that the assertion language L_A is expressive. To show completeness we assume that $\{P\}\ A\ \{Q\}$ is true and show that it is provable (given a complete proof system T for L_A relative to $\mathfrak{a}$). The proof will be by induction on the structure of A. The reader may verify that the only interesting case is when A has the form $C_3\ [b:\ A_1, A_2]$. We will split the proof of this case into two parts.

First we show that it is sufficient to know that

$$\{\bar{w} = \bar{w}_0\}\ X\ \{FX(\bar{w} = \bar{w}_0)\}$$

is always deducible where $\bar{w}$ are the variables which are active in the program and $\bar{w}_0$ are new non-active variables. Suppose $\{P\}\ X\ \{Q\}$ (i.e. $\{P\}\ C_3\ [b:\ A_1; A_2]\ \{Q\}$) is true so that $FX(P) \rightarrow Q$ holds. (Note that since L_A is expressive, $FX(P) \rightarrow Q$ is representable by a formula of L_A. In fact, if T is a complete proof system for L_A then $\vdash_T FX(P) \rightarrow Q$.) If $\{\bar{w} = \bar{w}_0\}\ X\ \{FX(\bar{w} = \bar{w}_0)\}$ is deducible, then $\vdash\{\bar{w} = \bar{w}_0 \wedge P(\bar{w}_0)\}\ X\ \{FX(\bar{w} = \bar{w}_0) \wedge P(\bar{w}_0)\}$ by axioms R1 and R2. Thus, $\vdash\{\exists \bar{w}_0\ [\bar{w} = \bar{w}_0 \wedge P(\bar{w}_0)]\}\ X\ \{\exists\ \bar{w}_0\ [FX(\bar{w} = \bar{w}_0) \wedge P(\bar{w}_0)]\}$ follows by application of R3. Hence

$$\vdash\{\exists \bar{w}_0\ [\bar{w} = \bar{w}_0 \wedge P(\bar{w}_0)]\}\ X\ \{FX(\exists \bar{w}_0\ [\bar{w} = \bar{w}_0 \wedge P(\bar{w}_0)])\}$$

by propositions 4.3.1 and 2.4.11 we conclude that $\{P(\bar{w})\}\ X\ \{FX(P(\bar{w}))\}$ is deducible and therefore by one additional application of the rule of consequence that $\{P\}\ X\ \{Q\}$ is also.

---

[1] It is also possible to prove soundness and completeness using the reverse predicate transformer R. In the proof of completeness we would use $\{RX\ (\bar{w} \neq \bar{w}_0)\}\ X\ \{\bar{w} \neq \bar{w}_0\}$ instead of $\{\bar{w} = \bar{w}_0\}\ X\ \{FX(\bar{w} = w_0)\}$. The argument based on $\{\bar{w} = \bar{w}_0\}\ X\ \{FX(\bar{w} = \bar{w}_0)\}$ seems considerably more natural however.

Next we show that $\{\bar{w} = \bar{w}_0\}$ X $\{FX(\bar{w} = \bar{w}_0)\}$ is indeed always deducible. To simplify notation, let

$$P = \{\bar{w} = \bar{w}_0\}$$

$$Q = FX(\bar{w} = \bar{w}_0)$$

$$T(\bar{w},\bar{w}_0) = FA_1(P \wedge b)$$

Note that all three of these predicates will be expressible by formulas of L_A.

$$FA_1(P \wedge b) = T(\bar{w}\ ,\bar{w}_0)$$

$$= \exists\ \bar{v}_0\ [\bar{w} = \bar{v}_0 \wedge T(\bar{v}_0,\bar{w}_0)]$$

$$= \exists\ \bar{v}_0\ [P\ \frac{\bar{v}_0}{\bar{w}_0} \wedge T(\bar{v}_0,\bar{w}_0)]$$

Hence by the induction assumption

$$\{P \wedge b\}\ A_1\ \{\exists\ \bar{v}_0\ [P\ \frac{\bar{v}_0}{\bar{w}_0} \wedge T(\bar{v}_0,\bar{w}_0)]\}$$

is deducible. Similarly we have

$$\exists\, v_0\ [Q\ \frac{\bar{v}_0}{\bar{w}_0} \wedge T(\bar{v}_0,\bar{w}_0)] = \exists\ \bar{v}_0\ [FX(\bar{w} = \bar{w}_0)\ \frac{\bar{v}_0}{\bar{w}_0} \wedge\ T(v_0,w_0)]$$

(Proposition 4.3.2) $= \exists\ \bar{v}_0\ [FX(\bar{w} = \bar{v}_0) \wedge T(v_0,w_0)]$

(Proposition 4.3.3) $= \exists\ \bar{v}_0\ [FX(\bar{w} = \bar{v}_0 \wedge T(v_0,w_0))]$

(Proposition 2.4.11) $= FX(\exists\, v_0[\bar{w} = \bar{v}_0 \wedge T(\bar{v}_0,\bar{w}_0)]\ )$

Hence $\exists\ \bar{v}_0\ [Q\ \frac{\bar{v}_0}{\bar{w}_0} \wedge T(\bar{v}_0,\bar{w}_0)] = FX(FA_1\ (P \wedge b))$

So, $FA_2\ (\exists \bar{v}_0\ [Q\ \frac{\bar{v}_0}{\bar{w}_0} \wedge T(\bar{v}_0,\bar{w}_0)]\ ) = FA_2\ (FX(FA_1(P \wedge b)))$

But by the predicate transformer fixed point theorem, we have

$Q = FX(\bar{w} = \bar{w}_0) = FA_2\ (FX(\ FA_1\ (P \wedge b))) \vee (\sim\ b \wedge P)$.

Thus, $FA_2\ (\exists \bar{v}_0\ [Q\ \frac{\bar{v}_0}{\bar{w}_0} \wedge T(\bar{v}_0,\bar{w}_0)]) \rightarrow Q$.

By the induction hypothesis, we have that

$$\exists \bar{v}_0\ [Q\ \frac{\bar{v}_0}{\bar{w}_0} \wedge T(\bar{v}_0,\bar{w}_0)]\ \ A_2\ \{Q\}$$

is deducible.

Since $P \wedge \sim b \rightarrow Q$ also follows from the predicate transformer fixed point theorem all of the hypothesis of (*) are satisfied and we may conclude that $\vdash\{P\}\ X\ \{Q\}$ (i.e. $\{P\}\ C_3\ [b:\ A_1;A_2]\ \{Q\}$ is is deducible where $P = \{\bar{w} = \bar{w}_0\}$ and $Q = \{FX(\bar{w} = \bar{w}_0)\}$). This completes the proof.

To illustrate how rules of inference of the kind described above may be used, we consider a simple example. Consider the procedure definition:

$$X = (n > 0) \rightarrow n := n - 2;\ X;\ n := n + 1,\ I$$

This may be considered an instance of the GCS $C_3\ [b:\ A_1;\ A_2]$ where b is $\{n > 0\}$, A_1 is $n := n - 2$, and A_2 is $n := n + 1$. We show that $\{P(n)\}\ X\ \{Q(n)\}$ is deducible where $P(n)$ is $4|n$ and $Q(n)$ is $2|n$. If we let

$$U = \{n = n_0\}$$

$$V = \{n = \frac{n_0}{2}\}$$

$$T = \{n_1 = n_0 - 2\}$$

Then

$$\{U \wedge b\}\ A_1\ \{\exists n_1\ [U \frac{n_1}{n_0} \wedge T]\}$$

$$\{\exists n_1 [V \frac{n_1}{n_0} \wedge T]\}\ A_2\ \{V\}$$

$$U \wedge \sim b \rightarrow V$$

all hold so that we may conclude $\{U\}\ X\ \{V\}$ also holds as an instance of the axiom given above. Since $P \rightarrow \exists n_0\ [U \wedge P(n_0)]$ and $\exists u_0\ [V \wedge P(n_0)] \rightarrow Q$ we may conclude $\vdash \{P\}\ X\ \{Q\}$ by using axioms R1, R2, and R3.

The crucial step in the completeness proof is showing that in order to prove the true assertion $\{P\}\ X\ \{Q\}$ it is sufficient to first prove $\{\bar{w} = \bar{w}_0\}\ X\ \{FX(\bar{w} = \bar{w}_0)\}$ where $\bar{w}$ are the variables active in X and $\bar{w}_0$ are new nonactive variables. This observation is originally due to Gorelick [GO75]. Although a simple algorithm can be given for generating sound and complete axioms for nonregular GCS's such as the one given for C_3 above, there is a much more natural way of organizing correctness proofs for nonregular procedures. This method will be discussed in the next section.

Although the use of non-regular systems of procedure declarations is common in programming, the reader may have noticed that all commonly used control structures are regular. This includes the "while" statement, the "repeat-until" statement, even the "Zahn" construct. One reason for this situation should be apparent from the discussion above--even simple non-regular control structures have complicated axioms. Statement patterns will be used frequently only if they are easy to understand, and control structures which are easy to understand will not have complicated axioms.

4.4 Gorelick's Theorem

In this section we give a more natural method for handling non-regular control structure (systems of recursive procedures). Instead of trying to find a different axiom for each distinct control structure, we give a set of five axioms that work for all control structures (systems of recursive procedures). The first three of these axioms are the axioms R1-R3 given in the previous section. In addition we will need

R4 $$\frac{\{P\}\ A\ \{Q\}}{\{P\sigma\}\ A\ \{Q\sigma\}}$$

where σ is an admissible substitution. This is the axiom of variable substitution given in Gorelick [G075].

R5 $$\frac{\{P\}\ Y\ \{Q\} \vdash \{P\}\ \tau(Y)\{Q\}\ Y,\ \text{a dummy procedure name}}{\{P\}\ Y\ \{Q\}}$$

where the procedure X has declaration $X \equiv \tau(X)$. This is the axiom of recursive invocation first given in [HO71].

The soundness of Axiom R4 follows immediately from proposition 4.3.3 Assume $\{P\}\ X\ \{Q\}$. Then $FX(P) \rightarrow Q$ and $FX(P)\sigma = FX(P\sigma)$ since σ is admissible. Thus

$$FX(P)\sigma \rightarrow Q\sigma$$

or

$$FX(P\sigma) \rightarrow Q\sigma$$

It follows that $\{P\sigma\}\ X\ \{Q\sigma\}$ must hold also and that the axiom is sound.

The soundness of axiom R5 follows directly from the fundamental invariance theorem. Assume that $\{P\}\ Y\ \{Q\} \vdash \{P\}\ \tau(Y)\ \{Q\}$ where Y is a dummy procedure name. Then if $FX^i(P) \rightarrow Q$ holds, we will have $\{P\}\ X^i\ \{Q\}$ and therefore by the assumption above

$$\{P\}\ \tau(X^i)\ \{Q\}$$

or

$$\{P\}\ X^{i+1}\ \{Q\} \quad \text{since} \quad X^{i+1} = \tau(X^i).$$

It follows that $FX^{i+1}(P) \rightarrow Q$ must also hold and hence by the fundamental invariance theorem that $P \rightarrow RX(Q)$ or $\{P\}\ X\ \{Q\}$ holds.

We call a procedure declaration $X \equiv \tau$ elementary if all of the procedure calls in τ are on the procedure X itself. If we restrict programs so that the only procedure declarations allowed are elementary declarations[1] then axioms R1 to R5 together with the axioms of the basic set also give completeness. The proof of this fact is originally due to Gorelick [G075], we include it here since it can be understood very easily with the tools that we have developed and since we will consider modifications of this result to total correctness and to programming languages with block structure, static scope and global variables in later sections.

We show that if the assertion language L_A is expressive (with respect to L_E and $\mathfrak{A}$) and if we are given a complete proof system T for L_A (relative to $\mathfrak{A}$) then $\{P\}\ A\ \{Q\}$ is true iff it is provable. The proof will be by induction on the structure of A and, as in the last section, the only interesting case is when A is a procedure call i.e. A is X where $X \equiv \tau$ occurs in the declaration part of the program. Again, by the argument of the last section we know that it is sufficient to show that $\{\bar{w} = \bar{w}_0\}\ X\ \{FX(\bar{w} = \bar{w}_0)\}$ is deducible where $\bar{w}$ are the variables active in X and $\bar{w}_0$ are new non-active variables.

---

[1] A more complicated version of R5 is needed to handle the general case of mutually recursive declarations [G075].

Let $P_0 = \{\bar{w} = \bar{w}_0\}$ and $Q_0 = \{FX(\bar{w} = \bar{w}_0)\}$, we will show that $\{P_0\}\ Y\ \{Q_0\} \vdash \{P_0\}\ \tau(Y)\ \{Q_0\}$. The desired conclusion $\{P_0\}\ X\ \{Q_0\}$ will then follow by R5. The proof of this result is also by a structural induction--this time on the structure of τ using an induction hypothesis which is slightly more general than what we really want to prove.

Induction hypothesis:

Let $\tau(X)$ be a statement, let P, Q be predicates then if $\{P\}\ \tau(X)\ \{Q\}$ is true, then

$$\{P_0\}\ Y\ \{Q_0\} \vdash \{P\}\ \tau(Y)\ \{Q\}$$

Cases on τ:

(i) τ is "I", "Ω", or "u := e": these cases are trivial and will be left to the reader

(ii) τ is $b \to A_1(X), A_2(X)$: if $\{P\}\ \tau(X)\ \{Q\}$ is true then $F[b \to A_1(X), A_2(X)]\ (P) \to Q$ is true, so,

$F[A_1(X)]\ (P \wedge b) \to Q$ i.e. $\{P \wedge b\}\ A_1(X)\ \{Q\}$

and

$F[A_2(X)]\ (P \wedge \sim b) \to Q$ i.e. $\{P \wedge \sim b\}\ A_2(X)\ \{Q\}$

are also true by properties of the predicate transformer R. By the induction hypothesis, we obtain

$$\{P_0\}\ Y\ \{Q_0\} \vdash \{P \wedge b\}\ A_1(Y)\ \{Q\}$$

and

$$\{P_0\}\ Y\ \{Q_0\} \vdash \{P \wedge \sim b\}\ A_2(Y)\ \{Q\}.$$

Combining and using the axiom for the conditional, we get

$$\{P_0\}\ Y\ \{Q_0\} \vdash \{P\}\ b \to A_1(Y), A_2(Y)\ \{Q\}$$

as required.

(iii) τ is $(A_1;A_2)$: proof is similar to (ii) above.

(iv) τ is X: $\{P\}\ X\ \{Q\}$ is true so $FX(P) \to Q$ is true. L_A is expressive so $FX(P) \to Q$ is representable by a formula of L_A and is provable in T. Thus from $\{\bar{w} = \bar{w}_0\}\ Y\ \{FX(\bar{w} = \bar{w}_0)\}$ we may derive

$\{\bar{w} = \bar{w}_0 \wedge P(\bar{w}_0)\}\ Y\ \{FX(\bar{w} = \bar{w}_0) \wedge P(\bar{w}_0)\}$ axioms R1, R2

$\{\bar{w} = \bar{w}_0 \wedge P(\bar{w}_0)\}\ Y\ \{FX(\bar{w} = \bar{w}_0 \wedge P(w_0))\}$ Prop. 4.3.2

$\{\exists \bar{w}_0\ [\bar{w}_0 \wedge P(\bar{w}_0)]\}\ Y\ \{\exists \bar{w}_0\ [FX(\bar{w} = \bar{w}_0 \wedge P(\bar{w}_0))\}$ axiom R3

$\{\exists \bar{w}_0\ [w = w_0 \wedge P(\bar{w}_0)]\}\ Y\ \{FX(\exists \bar{w}_0\ [\bar{w} = \bar{w}_0 \wedge P(\bar{w}_0)])\}$ Prop. 2.4.11

$\{P(\bar{w})\}\ Y\ \{FX(P(\bar{w}))\}$

$\{P\}\ Y\ \{Q\}$ rule of consequence

We illustrate the above axioms by a simple example. Suppose X has the declaration

$$X \equiv (n > 0) \to n := n - 1;\ X;\ m = m + 1;\ X;\ n := n + 1,\ I$$

we prove that $\{n = n_0 \wedge m = 0\}\ X\ \{n = n_0 \wedge m = 2^{n_0} - 1\}$

In order to establish this result it is first necessary to establish a somewhat more general result

$$\{n = n_0 \wedge m = m_0\}\ X\ \{n = n_0 \wedge m = 2^{n_0} - 1 + m_0\}$$

to prove this we assume $\{n = n_0 \wedge m = m_0\}\ Y\ \{n = n_0 \wedge m = 2^{n_0} - 1 + m\}$ and deduce

$$\{n = n_0 \wedge m = m_0\}\ \tau(Y)\ \{n = n_0 \wedge m = 2^{n_0} - 1 + m_0\}$$

where $\tau(Y) = (n > 0) \rightarrow n := n-1; Y; m := m+1; Y; n := n+1$, I. Clearly, we have $\{n = n_0 \wedge m = m_0 \wedge n = 0\}$ I $\{n = n_0 \wedge m = 2^{n_0} - 1+m_0\}$; thus we only need show

$\{n = n_0 \wedge m = m_0 \wedge n > 0\}$ $n := n-1$; Y; $m := m+1$; Y; $n := n+1$ $\{n = n_0 \wedge m = 2^{n_0} - 1+m_0\}$

This follows from the string of assertions below:

1) $\{n = n_0 \wedge m = m_0 \wedge n > 0\}$ $n := n-1$ $\{n = n_0-1 \wedge m = m_0\}$ Assignment

2) $\{n = n_0-1 \wedge m = m_0\}$ Y $\{n = n_0-1 \wedge m = 2^{n_0-1} - 1+m_0\}$ R_4: $\sigma = \frac{n_0-1}{n_0}$

3) $\{n = n_0-1 \wedge m = 2^{n_0-1} - 1+m_0\}$ $m := m+1$ $\{n = n_0-1 \wedge m = 2^{n_0-1} + m_0\}$ Assignment

4) $\{n = n_0-1 \wedge m = 2^{n_0-1} + m_0\}$ Y $\{n = n_0-1 \wedge m = 2^{n_0} - 1+m_0\}$ R4: $= \frac{n_0-1}{n_0} \frac{2^{n_0} + m_0}{m_0}$

5) $\{n = n_0 -1 \wedge m = 2^{n_0} - 1+m_0\}$ $n := n+1$ $\{n = n_0 \wedge m = 2^{n_0} - 1+m_0\}$ Assignment

If 1 - 5 above are combined using the rule of consequence, the desired conclusion $\{n = n_0 \wedge m = m_0 \wedge n > 0\}$ $n := n-1$; Y; $m := m+1$; Y; $n := n+1$: $\{n = n_0 \wedge m = 2^{n_0} - 1 + m\}$, is obtained. Note that in applying R4 to a precondition - postcondition expression which involves a dummy procedure name Y one must use the real procedure declaration corresponding to Y (e.g. $X \equiv \tau(X)$) in determining whether variables are active or inactive in Y.

4.5 Total Correctness

In Chapter 3 and previous sections of Chapter 4 we have dealt exclusively with partial correctness. We have worked with assertions of the form {P} A {Q} which were considered true iff when P was true initially and A was executed, then either the execution of A did not terminate properly[1] or else Q was satisfied by the final state of the program. Unfortunately failure to terminate properly is one of the most frequent sources of error in incorrect programs. What we are really interested in is total correctness--we want to be able to make assertions of the form {P} A {Q} which will be true iff when P is true initially, A is guaranteed to terminate, and Q will hold of the final program state. Traditionally, proofs of total correctness have been split into two parts: (a) a proof of partial correctness and (b) a separate proof that the program terminates in a satisfactory manner. In this section we argue that such a division is unnecessary and that total correctness is, in a certain sense, no more difficult to establish than partial correctness. We show that it is possible to give a set of axioms for total correctness which is both sound and complete under an expressibility condition on the assertion language which is no more restrictive than that used in the proofs of soundness and completeness for partial correctness axioms.

We begin by giving total correctness axioms for the null statement, the assignment statement, the composition of statements, the conditional statement, and the rule of consequence. Since none of these statements

---

[1] Note that there are two ways in which a program can fail to terminate properly (i) by the occurence of an infinite loop of some sort and (ii) by the occurence of an abnormal condition such as division by zero or integer overflow. In this thesis we will not treat the latter case.

is capable of causing non-termination by itself, the axioms given are analogous to those of the basic set for partial correctness (see section 4.4).

1. $\langle P \rangle\ I\ \langle P \rangle$ — null statement

2. $\langle P\ \frac{e}{x} \rangle\ x := e\ \langle P \rangle$ — assignment

3. $\dfrac{\langle P \rangle\ A_1\ \langle Q \rangle,\ \langle Q \rangle\ A_2\ \langle R \rangle}{\langle P \rangle\ (A_1;A_2)\ \langle R \rangle}$ — composition

4. $\dfrac{\langle P \wedge b \rangle\ A_1\ \langle Q \rangle,\ \langle P \wedge \neg b \rangle\ A_2\ \langle Q \rangle}{\langle P \rangle (b \rightarrow A_1, A_2) \langle Q \rangle}$ — conditional

5. $\dfrac{P \rightarrow Q,\ \langle Q \rangle\ A\ \langle R \rangle,\ R \rightarrow S}{\langle P \rangle\ A\ \langle S \rangle}$ — consequence

Next we need total correctness axioms analogous to R1-R5 to handle recursive procedures. The first four of these axioms are very similar to the original partial correctness axioms.

6. $\langle T \rangle\ A\ \langle T \rangle$ — provided that T does not contain any free variables which are active in A.

7. $\dfrac{\langle P_1 \rangle\ A\ \langle Q_1 \rangle,\ \langle P_2 \rangle\ A\ \langle Q_2 \rangle}{\langle P_1 \wedge Q_1 \rangle\ A\ \langle P_2 \wedge Q_2 \rangle}$

8. $\dfrac{\langle P(u_0) \rangle\ A\ \langle Q(u_0) \rangle}{\langle \exists u_0\ P(u_0) \rangle\ A\ \langle \exists u_0\ Q(u_0) \rangle}$ — provided that u_0 is not active in A.

9. $\dfrac{\langle P \rangle\ A\ \langle Q \rangle}{\langle P\sigma \rangle\ A\ \langle Q\sigma \rangle}$ — provided that $\{P_\sigma\}\ A\ \{Q_\sigma\}$ is an admissible substitution.

The last axiom (the one analogous to R5) is the crucial axiom for total correctness-it allows proofs of total correctness to be constructed which use induction on the depth of recursion of a recursive procedure.

$$10.\ \frac{\langle P(0)\rangle\ \tau(\Omega)\ \langle Q\rangle, \langle P(i)\rangle\ r\ \langle Q\rangle \vdash \langle P(i+1)\rangle\ \tau(r)\ \langle Q\rangle}{\langle \exists\, i\ P(i)\rangle\ X\ \langle Q\rangle}$$

where the procedure X has declaration $X \equiv \tau(X)$.[1]

We illustrate the use of the axioms stated above by proving that $\langle n = n_0 \wedge m = m_0\rangle\ X\ \langle n = 0 \wedge m = 2^{n_0}\rangle$ is true where X is defined by

$X \equiv (n = 0 \rightarrow m := 1,$

$(\text{even}(n) \rightarrow n := n/2;\ X;\ m = m^2,$

$n := (n-1)/2;\ X;\ m = 2m^2))$

and even(n) is the predicate which is true if n is an even non negative integer. In this case $p(i) \equiv \{n = n_0 \wedge n_0 \leq i \wedge m = m_0\}$, and $Q \equiv \{n = 0 \wedge m = 2^{n_0}\}$. Note that $P \equiv \{n = n_0 \wedge m = m_0\}$ is true iff some $P(i)$ is true. It is also easy to show that $\langle P(0)\rangle \tau\ (\Omega)\ \langle Q\rangle$ is true. Thus to complete the proof we assume that $\langle P(i)\rangle\ r\ \langle Q\rangle$ is true and show that $\langle P(i+1)\rangle\ \tau(r)\ \langle Q\rangle$ must hold. This follows by the rule for composition of statements and the rule for the conditional from the chain of six assertions below.

(i) $\langle \text{even}(n) \wedge n = n_0 \wedge 0 < n_0 \leq i+1 \wedge m = m_0\rangle\ \ n := n/2\ \langle n = n_0 \wedge n_0 \leq i\ i \wedge m = m_0 \wedge n = 2k_0\rangle$

---

[1] This axiom is a variant of one originally suggested to the author by M. O'Donnell. Note that it assumes that the integers are a subset of the domain of the interpretation. If the domain is finite, a different rule using the same basic idea (induction on depth of recursion) gives completeness.

(ii) $\langle n = k_0 \wedge k_0 \leq i \wedge m = m_0 \wedge n_0 = 2k_0 \rangle \; r \; \langle n = 0 \wedge n_0 = 2k_0 \wedge m = 2^{k_0} \rangle$

(iii) $\langle n = 0 \wedge n_0 = 2k_0 \wedge m = 2^{k_0} \rangle \; m := m^2 \; \langle n = 0 \wedge m = 2^{n_0} \rangle$

(iv) $\langle \sim \text{even}(n) \wedge n = n_0 \wedge 0 < n_0 \leq i + 1 \wedge m = m_0 \rangle \; n := (n-1)/2 \langle n = k_0 \wedge k_0 \leq i \wedge m = m_0 \wedge n_0 = 2k_0 + 1 \rangle$

(v) $\langle n = k_0 \wedge k_0 \leq i \wedge m = m_0 \wedge n = 2k_0 + 1 \rangle \; r \; \langle n = 0 \wedge n_0 = 2k_0 + 1 \wedge m = 2^{k_0} \rangle$

(vi) $\langle n = 0 \wedge n_0 = 2k_0 + 1 \wedge m = 2^{k_0} \rangle \; m := 2m^2 \; \langle n = 0 \wedge m = 2^{n_0} \rangle$

Steps (i), (iii) (iv), and (vi) are straight forward. Steps (ii) and (v) involve using the assumption $\langle P(i) \rangle \; r \; \langle Q \rangle$ together with axioms 6,7, and 9. Thus, for example, we obtain (ii) as follows

$\langle n = n_0 \wedge n_0 \leq i \wedge m = m_0 \rangle \; r \; \langle n = 0 \wedge m = 2^{n_0} \rangle$ (by inductive assumption)

$\langle n = k_0 \wedge k_0 \leq i \wedge m = m_0 \rangle \; r \; \langle n = 0 \wedge m = 2^{k_0} \rangle$ (axiom 9 with $\sigma = \frac{k_0}{n_0}$)

$\langle n_0 = 2k_0 \rangle \; r \; \langle n_0 = 2k_0 \rangle$ (axiom 6)

$\langle n = k_0 \wedge k_0 \leq i \wedge m = m_0 \wedge n_0 = 2k_0 \rangle \; r \; \langle n = 0 \wedge n_0 = 2k_0 \wedge m = 2^{k_0} \rangle$ (axiom 7)

The rest of the proof is left to the reader.

The soundness of axioms 1-5 follows immediately from the results of Chapter 2. Axioms 6-9 may be proved sound by an argument similar to that used in showing the soundness of R1-R4. To simplify the soundness proof for axiom 10 we first prove the following lemma:

Lemma:

Let $\{P_i\} i \geq 0$ be a family of predicates. Suppose that $P_0 \rightarrow BX^1(Q)$ is true with respect to $\mathfrak{A}$ and that if $P_i \rightarrow BX^{i+1}(Q)$ is true then $P_{i+1} \rightarrow BX^{i+2}(Q)$ is true also. Under these conditions we may conclude that for all $n \geq 0$ $P_n \rightarrow BX(Q)$ holds.

Proof: As in the proof of the fundamental invariance theorem (see section 2.5) we translate the problem into set notation. From the hypothesis of the lemma we may conclude that $P_n \subseteq BX^{n+1}(Q)$ for all $n \geq 0$. Since $BX^n(Q) \subseteq BX(Q)$ the desired result follows immediately.

To establish the soundness of axiom 10, we assume that $\langle P(0) \rangle\ \tau(\Omega)\ \langle Q \rangle$ and $\langle P(i) \rangle\ r\ \langle Q \rangle \vdash \langle P(i+1) \rangle\ \tau(r)\ \langle Q \rangle$ are both true. Let $P_i = P(i)$. Since $\langle P(0) \rangle\ \tau(\Omega)\ \langle Q \rangle$ is true, we see that $P_0 \rightarrow BX^1(Q)$ is true. Since $\langle P(i) \rangle\ r\ \langle Q \rangle \vdash \langle P(i+1) \rangle\ \tau(r)\ \langle Q \rangle$ holds and $X^{i+2} = \tau(X^i)$, we see that if $P_i \rightarrow BX^i(Q)$ is true $P_{i+1} \rightarrow BX^{i+2}(Q)$ must be true also. By the above lemma $P_n \rightarrow BX(Q)$ holds for all $n \geq 0$. Since $\exists i\ P(i)$ is true if and only if some P_i is true, we see that $\exists\ i\ P(i) \rightarrow BX(Q)$ is true or equivalently that $\langle \exists i\ P(i) \rangle\ X\ \langle Q \rangle$ is true.

We next turn to the question of completeness. Since the inductive argument used to prove completness in the total correctness case is almost identical in structure to that given in the previous section for partial correctness, we will merely try to indicate the ideas involved--not give a full proof. Let D be the domain of $\mathfrak{A}$. We will assume that:

a) both L_A and L_E are extensions of L_N, the language of number theory, and that the symbols of $L_A(L_E)$ which are common to L_N receive their standard interpretation under $\mathfrak{A}$, i.e. $N \subseteq D$.

b) there is a predicate "integer (X)" defined on D which is true of X iff X is an integer (this provides a means of quantifying over the integers).

c) for all statements A and formulas Q in L_A there is a formula in L_A which expresses $BA(Q)$.

Note that in c) we could just have easily required that $RA(Q)$ or $FA(Q)$ be expressible (see Theorem 3.4.6). Thus, if we restrict our attention to interpretations which already satisfy a) and b) we will not be requiring a stronger notion of expressibility in the total correctness case than was used in the partial correctness case.

Let T be a complete proof system for L_A relative to $\mathfrak{A}$. Assume that $\langle U \rangle\ A\ \langle V \rangle$ is true with respect to $\mathfrak{A}$. We show that $\langle U \rangle\ A\ \langle V \rangle$ is provable using axioms 1-10 above and the proof system T. The proof is by induction on the structure of A. Since the proof is similar to the one given in section 4.4, we will only consider the case where A is a call on the X which has a declaration of the form $X \equiv (b \rightarrow A_1;\ X;\ A_2,\ I)$. We will show

(i) $\langle U \rangle\ X\ \langle V \rangle$ is provable if $\langle P \rangle\ X\ \langle Q \rangle$ is provable where $P \equiv BX(\bar{w} = \bar{w}_0)$ and $Q \equiv (\bar{w} = \bar{w}_0)$ ($\bar{w}$ is the vector of variables which is active in X and $\bar{w}_0$ consists of new non-active variables).

(ii) There is a predicate $P(i)$ (expressible in L_A) such that

a) P is true iff $P(i)$ is true for some value of i.

b) $\langle P(0) \rangle\ \tau(\Omega)\ \langle Q \rangle$ is deducible.

c) $\langle P(i) \rangle\ r\ \langle Q \rangle \vdash \langle P(i+1) \rangle\ \tau(r)\ \langle Q \rangle$ holds.

Proof of (i):

By assumption $\langle BX(\bar{w} = \bar{w}_0)\rangle \times \langle \bar{w} = \bar{w}_0\rangle$

By axioms 6, 7 $\langle BX(\bar{w} = \bar{w}_0) \wedge V(\bar{w}_0)\rangle \times \langle \bar{w} = \bar{w}_0 \wedge V(\bar{w}_0)\rangle$

Hence $\langle BX(\bar{w} = \bar{w}_0 \wedge V(\bar{w}_0))\rangle \times \langle \bar{w} = \bar{w}_0 \wedge V(\bar{w}_0)\rangle$

By axiom 8 $\langle \exists \bar{w}_0\, BX(\bar{w} = \bar{w}_0 \wedge V(\bar{w}_0))\rangle \times \langle \exists \bar{w}_0[\bar{w} = \bar{w}_0 \wedge V(\bar{w}_0)]\rangle$

$\langle BX(\exists w_0[w = w_0 \wedge V(w_0)])\rangle \times \langle \exists w_0[w = w_0 \wedge V(w_0)]\rangle$

Since $V \equiv \exists \bar{w}_0[\bar{w} = \bar{w}_0 \wedge V(w_0)]$ and $\vdash U \rightarrow BX(V)$, we get that $\langle U\rangle \times \langle V\rangle$ as required.

Proof of (ii):

Choose $P(i) \equiv BX^{i+1}(\bar{w} = \bar{w}_0)$. It is straightforward to show that $P(i)$ is expressible by a formula of L_A and that P is true iff some $P(i)$ is true.

Since $P(0) = BX^1(\bar{w} = \bar{w}_0)$ and $X^1 = \tau(\Omega)$ we see that $\vdash \langle P(0)\rangle\ \tau(\Omega)\ \langle Q\rangle$. Finally we assume that $\vdash \langle P(i)\rangle\ r\ \langle Q\rangle$ holds and show that $\vdash \langle P(i+1)\rangle\ \tau(r)\ \langle Q\rangle$ must hold also. Let $U(\bar{w},\bar{w}_0) = BA_2(\bar{w} = \bar{w}_0) = BA_2(Q)$, then $BA_2(Q) = \exists \bar{w}_1[Q\sigma \wedge T]$ where $T = U(\bar{w}_1,\bar{w}_0)$ and $\sigma = \frac{\bar{w}_1}{\bar{w}_0}$, $P(i+1) \wedge b = BA_1(\exists \bar{w}_1[P(i)\sigma \wedge T])$, and $P(i+1) \wedge \sim b \rightarrow Q(i+1)$. From $BA_2(Q) = \exists \bar{w}_1[Q\sigma \wedge T]$ we get $\vdash \langle \exists \bar{w}_1[Q\sigma \wedge T]\rangle\ A_2\ \langle Q\rangle$. By induction we get $\vdash \langle \exists \bar{w}_1[Q\sigma \wedge T]\rangle\ A_2\ \langle Q\rangle$. From $P(i+1) \wedge b = BA_1(\exists \bar{w}_1[P(i)\sigma \wedge T])$ we get $\vdash \langle P(i+1) \wedge b\rangle\ A_1\ \langle \exists \bar{w}_1 P(i)\sigma \wedge T]\rangle$. By induction $\vdash \langle P(i+1) \wedge b\rangle\ A_1\ \langle \exists \bar{w}_1[P(i)\sigma \wedge T]\rangle$. Thus by assumption we have $\vdash \langle P(i)\rangle\ r\ \langle Q\rangle$; by axiom 9, $\vdash \langle P(i)\sigma\rangle\ r\ \langle Q\sigma\rangle$; by axioms 6 and 7, $\vdash \langle \exists \bar{w}_1[P(i)\sigma \wedge T]\rangle\ r\ \langle \exists \bar{w}_1[Q\sigma \wedge T]\rangle$; by rule of consequence, $\vdash \langle P(i+1) \wedge b\rangle\ A_1; r; A_2\ \langle Q\rangle$; by rule for conditional, $\vdash \langle P(i+1)\rangle\ (b \rightarrow A_1; r; A_2; I)\ \langle Q\rangle$ or $\vdash \langle P(i+1)\rangle\ \tau(r)\ \langle Q\rangle$.

Chapter 5

PROGRAMMING LANGUAGE CONSTRUCTS FOR WHICH IT IS IMPOSSIBLE TO OBTAIN GOOD HOARE-LIKE AXIOMS

5.1 Introduction

It is well known that Hoare-like deduction systems for establishing partial correctness of programs may fail to be complete because of a) incompleteness of the assertion language relative to the underlying interpretation and b) the inability of the assertion language to express the invariants of loops. S. Cook [CO75] has shown that if these two sources of incompleteness are taken into account (i.e., by using an "expressive" assertion language together with a complete proof system for the assertion language) then sound, complete axiom systems for a fairly large fragment of Algol may be devised.

We show that there are natural control mechanisms for which it is impossible to obtain sound, complete sets of Hoare-like axioms even in this special sense of Cook's. While such incompleteness is expected with data structures (e.g. the integers, stacks, queues, etc.), it is new and somewhat surprising that it should exist for control mechanisms. These results suggest that such constructs will be difficult to prove correct and (one might argue) should be avoided in the design of languages suitable for program verification.

The first such programming language feature considered is recursive procedures with procedure parameters in the presence of global variables and static scope. This result is surprising since it holds even if we disallow calls of the form Call P(...,P)* and since it is possible to obtain

*Calls of the form "Call P(...,P)" appear to be necessary if one wants to directly simulate the lambda-calculus by passing procedure parameters.

a complete Hoare-like axiom system in the presence of static scope and global variables if we either a) allow recursive procedures with variable parameters (call by reference) but disallow procedure parameters or b) allow procedure parameters but require that procedures be non-recursive.

Our discussion of b) appears to be the first explicit treatment of how one can handle static scope and global variables by means of Hoare-like axioms. Cook (C075) and Gorelick (G075) both discuss global variables but the semantics of the programming language that they consider assumes dynamic scope. Our proof that it is impossible to obtain a sound, complete set of axioms for the full language (i.e., without restrictions such as a) or b) uses a technique developed by Jones and Muchnick [J075] in which a queue machine with undecidable halting problem is simulated in the runtime stack.

The second feature that we consider is coroutines. If procedures are not allowed to be recursive then there is a simple axiomatic method for proving correctness of coroutines which can be shown to be complete. This method is based on adding auxilliary variables which serve as program counters see [OW76]). If procedures are allowed to be recursive then it is shown that no such simple method can give completeness. These observations generalize in a natural manner to languages with parallelism and recursion.

Similar arguments are also applicable to a number of other programming language features including (a) call by name with functions and global variables, (b) pointer variables with retention, (c) pointer variables with recursion, and (d) label variables with retention. All of the features described above can be viewed as being too complicated for a simple axiomatic description of the type advocated by Hoare [H071 and H073] and

thus, in some sense, inherently difficult to prove correct.

A number of open problems are stated in section 5.9.

5.2 A Simple Programming Language and its Semantics.

As in [CO74] we distinguish two logical systems involved in discussions of program correctness--the assertion language L_A in which predicates describing a program's behavior are described and the expression language L_E in which the right hand sides of assignment statements and the booleans of conditionals and while statements are specified. Both L_A and L_E are assumed to be first order languages with equality and L_A is an extension of L_E. Thus, in particular, the variables of L_E will be a proper subset of the variables of L_A. The variables of L_E are called program identifiers (PROG_ID) and are assumed to be ordered by the positive integers. The variables of L_A are called variable identifiers (VAR_ID). Let $\mathfrak{a}$ be an interpretation for L_A. Once $\mathfrak{a}$ has been specified, meanings may be assigned to the terms and formulas of $L_A(L_E)$ using the standard technique of first order model theory.

A state is a mapping from a finite subset of VAR_ID to D where D is the domain of the interpretation $\mathfrak{a}$.[1] If s is a state, i a variable identifier, and d an element of D, then $s[i \leftarrow d]$ is the function with domain $DOM(s) \cup \{i\}$ which is given by

$$s[i \leftarrow d](v) = \begin{cases} d & \text{if } v = i \\ s(v) & \text{if } v \neq i,\ v \in \text{VAR\_ID} \end{cases}$$

Similarly, if $A \subseteq DOM(s)$, then $s|_A$ is the function with domain A which is given by

[1] Note that the definition of state given above is different from that used earlier in the paper.

$$s|_A(v) = s(v) \quad \text{for} \quad v \in A$$

In particular DEL(s,i) is the function with domain DOM(s) - {i} given by

$$DEL(s,i) = s|_{DOM(s)-\{i\}}$$

If P is a formula of the assertion language L_A with free variables $x_1, x_2, \ldots, x_n$ then we will use the notation P(s) to mean $P^{s(x_1),\ldots,s(x_n)}_{x_1,\ldots,x_n}$

An environment π is a mapping from a finite subset of PROC_ID to FORMAL_PARAMETER_LISTS X STMTS. Informally $\pi(q) = \langle(\overline{x};\overline{p}), K\rangle$ means that the procedure with name q has declaration "q:proc $(\overline{x}:\overline{p})$; K end" where $\overline{x}$ are the formal variable parameters and $\overline{p}$ are the formal procedure parameters. The notation $\pi[q \leftarrow \langle(\overline{x}:\overline{p}), K\rangle]$ is defined in a manner similar to that given above for $s[i \leftarrow a]$ and should be self explanatory.

The meaning function $M = M_a$ associates with a statement A, state s and environment π a new state s'. Intuitively s' is the state resulting if A is executed with initial state s and intitial environment π. The definition of M is given operationally in a rather non-standard manner which makes extensive use of renaming. This type of definition has the following two advantages:

a) It is very close to the original statement of the copy rule in the algol 60 report [NA63] -- thus there should be no question that we are using static scope.

b) It simplifies the proof of soundness and completeness for the Hoare-like axioms given in later sections. The definition of

$M[A](s)(\pi)$ is by cases on A:

(1) A is "begin new x; B end" $\rightarrow$ $DEL(M[\text{begin } B \frac{x^i}{x} \text{ end}](s')(\pi), x^i)$ where i is the index of the first program identifier not appearing in A, π, or $DOM(s)$ and $s' = s[x^i \leftarrow e_0]$. ($e_0$ is a special domain element which is used as the initial value of program identifiers.)

(2) A is "begin q: proc$(\bar{x}:\bar{p})$; K end; B end" $\rightarrow$ $M[\text{begin } B \frac{q^i}{q} \text{ end}](s)(\pi')$ where i is the index of the first procedure identifier not occuring in A or π and $\pi' = \pi[q^i \leftarrow \langle(x:p), K\frac{q^i}{q}\rangle]$. Note that we are assuming that the syntax of allowable programs has been restricted to require that procedures be declared before they are used.

(3) A is "begin B_1; B_2 end" $\rightarrow$ $M[\text{begin } B_2 \text{ end}](M[B_1](a)(\pi))(\pi)$"

(4) A is "begin end" $\rightarrow$ s

(5) A is "x := e" $\rightarrow$ s' where $s' = s[x \leftarrow \mathcal{A}[e(s)]]$

(6) A is "b $\rightarrow$ A_1,A_2" $\rightarrow$ $\begin{cases} M[A_1](s)(\pi) & \text{if } \mathcal{B}[b(s)] = \text{true} \\ M[A_2](s)(\pi) & \text{o.w.} \end{cases}$

(7) A is "b*A" $\rightarrow$ $\begin{cases} M[A; B*A](s)(\pi) & \text{if } \mathcal{B}[b(s)] = \text{true.} \\ s & \text{o.w.} \end{cases}$

("b*A" is our short-hand notation for the statement "while b do A")

(8) A is "call $q(\bar{a}: \bar{P})$" $\rightarrow$ $M[K\frac{\bar{a}}{\bar{x}}\frac{\bar{P}}{\bar{p}}](s)(\pi)$ where $\pi(q) = \langle(\bar{x}:\bar{p}), K\rangle$. Here $\bar{a}$ is the list of actual variable parameters and $\bar{P}$ is the list of actual procedure parameters. Note that the entries in $\bar{a}$ must be simple program identifiers.

If P is a formula of the assertion language L_A which has free variables $x_1, x_2, \ldots, x_n$, then we identify P with the set of all those states s with domain $\{x_1, x_2, \ldots, x_n\}$ such that if $s(x_i) = c_i$ for $1 \leq i \leq n$ then $P\frac{c_1, \ldots, c_n}{x_1, \ldots, x_n}$. By convention the predicate TRUE is identified

with the set containing the "empty" function $s: \phi \to D$, while the predicate FALSE is identified with the empty state set. Under these conventions note that the implication $P \to Q$ will be true with respect to $\mathfrak{a}$ if

$$\forall s[s \in P \Rightarrow s|_{free(Q)} \in Q]$$

where $free(Q)$ is the set of program identifiers which are free in Q.

We will be concerned with partial-correctness assertions of the form $\{P\}\ A\ \{Q\}\ /D$ where A is a program statement, P, Q are formulas of L_A and D is a set of procedure declarations.

5.2.1 Definition:

We say that $\{P\}\ A\ \{Q\}\ /D$ is true with respect to $\mathfrak{a}$ iff

$$\forall s, s'[s \in P \wedge M[A](s)(\pi) = s' \Rightarrow s'|_{free(Q)} \in Q] \text{ where}$$

π is the environment corresponding to D, i.e. $\pi(q) = \langle (\bar{x}:\bar{p}),\ K \rangle$ iff "q: proc $(\bar{x}:\bar{p})$; K end" $\in D$.

This is the usual definition of partial correctness. If $\{P\}\ A\ \{Q\}\ /D$ is true with respect to $\mathfrak{a}$, we will write $\models_{\mathfrak{a}} \{P\}\ A\ \{Q\}/D$. Note that in order for $\models_{\mathfrak{a}} \{P\}\ A\ \{Q\}/D$ we are implicitly requiring that free $(Q) \subseteq$ free (P). The program identifiers which are global to A or to some procedure in D must also comprise a subset of free(P).

5.2.2 Definition:

We say that L_A is expressive with respect to L_E and $\mathfrak{a}$ iff for all A, Q, π there is a formula of L_A which expresses $R[A]\ (\pi)\ (Q)$

(i.e. $F[A]\uparrow(\pi)(Q)$) where

$R[A](\pi)(Q) = \{s \mid$ DOM(s) consists of those variables which are free in Q or global to A or some procedure in π and $M[A](s)(\pi)\uparrow$ or $M[A](s)(\pi)\big|_{free(Q)} \in Q\}$

and

$F[A](\pi)(Q) = \{M[A](\pi)(s) \mid s \in Q\}$.

5.2.3 Proposition:

Suppose that L_A is expressive relative to L_E and $\hat{a}$, let π be the environment corresponding to D, then

(a) $\vdash_{\hat{a}} \{R[A](\pi)(Q)\}\ A\ \{Q\}/D$

(b) $\vdash_{\hat{a}} \{P\}\ A\ \{Q\}/D \Rightarrow \vdash_{\hat{a}} P \to R[A](\pi)(Q)$

Proof:

(a) $s \in R[A](\pi)(Q)$ and $M[A](s)(\pi) = s'$ implies that $s'\big|_{free(Q)} \in Q$.

(b) ($\Longrightarrow$) Suppose that $\{P\}\ A\ \{Q\}/D$ is true, then $s \in P$ implies that either $M[A](s)(\pi)\uparrow$ or $M[A](s)(\pi)\big|_{free(Q)} \in Q$. Hence $s \in P$ implies $s\big|_{free(R[A](\pi)(Q))} \in R[A](\pi)(Q)$.

Thus $\vdash_{\hat{a}} P \to R[A](\pi)(Q)$.

($\Longleftarrow$) $s \in P$ implies $s\big|_{free(R[A](\pi)(Q))} \in R[A](\pi)(Q)$.

Thus if $s \in P$ and $s' = M[A](s)(\pi)$ then $s'\big|_{free(Q)} \in Q$.

Hence $\vdash_{\hat{a}} \{P\}\ A\ \{Q\}/D$.

Thus, we see that $R[A](\pi)(Q)$ is the weakest precondition corresponding to A, π, and Q. Similarly, we may prove that $F[A](\pi)(Q)$ is the strongest post condition corresponding to A, π, and Q. Note that it is relatively easy to come up with examples of situations in which the assertion

language L_A is not expressive with respect to the expression language L_E and $\mathfrak{a}$. Cook [CO75] gives the example in which the assertion language L_A and the expression language are both the language of Presburger Arithmetic. Mitchell Wand [WA76] gives a different example of the same phenomenon.

Fortunately, more realistic choices for L_A, L_E, and $\mathfrak{a}$ do give expressibility. If L_A and L_E are both the full language of number theory, and $\mathfrak{a}$ is an interpretation in which the symbols of number theory receive their usual meanings, then L_A is expressive with respect to L_E and $\mathfrak{a}$. Also, if the domain of $\mathfrak{a}$ is finite, then we have expressibility.

5.2.4 Proposition:

If L_A, L_E are first order languages with equality and the domain of $\mathfrak{a}$ is a finite set, then L_A will be expressive with respect to L_E and $\mathfrak{a}$.

Proof: Let D be the domain of $\mathfrak{a}$ and suppose that $|D| < \infty$. Let A, Q, and π be given. Suppose that $x_1,\ldots,x_n$ are the variables which occur free in A, Q, π. Because of the finiteness of D there are only finitely many (say m) n-tuples $\langle a_1^j \ldots a_n^j \rangle$ such that if we define $s_j(x_i) = a_i^j$ then either $M[A](s_j(\pi))\uparrow$ or $M[A](s_j)(\pi)\big|_{free(Q)} \in Q$.

Let $R = \bigvee_{1 \leq j \leq m} x_1 = a_1^j \wedge x_2 = a_2^j \wedge \ldots \wedge x_n = a_n^j$, then it is not difficult to show that R expresses $R[A](\pi)(Q)$.

5.3. Hoare-like Axioms for Static Scope, Global Variables, etc.

In this section we give a deductive system for handling

(a) static scope, (b) global variables, and (c) nonrecursive procedures with procedure parameters which is both sound and (in the sense of Cook) complete.

(H1) $$\frac{\{U \wedge x^i = e_0\} \text{ begin } A \frac{x^i}{x} \text{ end } \{V\}/D}{\{U\} \text{ begin new } x; A \text{ end } \{V\}/D}$$

where i is the index of the first program identifier not appearing in A, D, or P.

(H2) $$\frac{\{U\} \text{ begin } A \frac{q^i}{q} \text{ end } \{V\}/D \cup \{q^i: \text{proc } (\bar{z}:\bar{p}); K \frac{q^i}{q} \text{ end}\}}{\{U\} \text{ begin } q: \text{proc } (\bar{z}:\bar{p}); K; \text{ end}; A; \text{ end } \{V\}/D}$$

where i is the index of the first procedure identifier not appearing in K, A, or D.

(H3) $$\frac{\{U\} A \{V\}/D_1}{\{U\} A \{V\}/D_2}$$

provided that $D_1 \subseteq D_2$ and D_2 does not contain the declarations of two different procedures with the same name.

(H4) (a) $$\frac{\{U\} A \{V\}/D}{\{U\} \text{ begin } A \text{ end } \{V\}/D}$$

(b) $$\frac{\{U\} A_1 \{V\}/D, \{V\} \text{ begin } A_2 \text{ end } \{W\}/D}{\{U\} \text{ begin } A_1; A_2 \text{ end } \{W\}/D}$$

(H5) - (H8) Usual axioms for assignment, conditional, while, and consequence (see [HO71]). Note of course that each of these axioms must be modified to make explicit the set D of procedure declarations).

(H9) $$\frac{\{U\} K \frac{\bar{a}}{\bar{x}} \frac{\bar{P}}{\bar{p}} \{V\}/D \text{ which includes } \bar{P}}{\{U\} \text{ call } h(\bar{a}:\bar{p}) \{V\}/D \cup \{h: \text{proc } (\bar{x}:\bar{p}); K \text{ end}\}}$$

provided that D does not already contain a procedure declaration "h: $(\bar{x}:\bar{p}')$; K' end" different from "h: proc $(\bar{x}:\bar{p})$; K end".

Note that axiom H9 is extremely simple; it merely states that if we can prove the body of the procedure correct when we substitute the actual parameters for the formal parameters then we may conclude that the procedure call is correct. We illustrate how these axioms may be used to handle static scope by considering a simple example involving nonrecursive procedures with procedure parameters.

Example. {true}

```
begin
    h: proc(:p); begin new  x; x := 1;  call p(z); end
    f: proc( );
       begin new x;
           g: proc(y); y := x;  end
           x := 2;
           call  h(g);
           end;
       end;
    z := 3;
    call  f( );
    end;
{z = 2}/ ϕ
```

The reader should verify that this preconditon -- post condition assertion is correct with respect to the semantics given in section 2 (static scope). Note also that if dynamic scope is used, the correct post condition is z = 1.

Proof:

(1) $\{x' = 2\}\ z := x'\ \{z = 2\}/\phi$ H5

(2) $\{x' = 2\}$ begin call g'(z) end $\{z = 2\}/$ {g': proc(y); y := x'; end} H9,H4a

(3) {x' = 2 ∧ x" = e_0} begin x" := 1; call g'(z); end {z = 2}/

{ g: proc;...x'...end} H5, H4b

(4) {x' = 2} begin new x x := 1; call g'(z); end {z = 2}/

{g: proc...x'...end} H1

(5) {x' = 2} call h(:g) {z = 2}/

{g: proc...x'...end, h: proc...end} H9

(6) {x' = e_0} begin x' := 2; call H(:g) end {z = 2}/

{g: proc...x'...end, h: proc...end} H5, H4a, b

(7) {x' = e_0} begin

g: proc(y); y := x'; end;

x' := 2;

call h(:g);

end; H2

{z = 2} / {h: proc...end}

(8) {true} begin

new x;

g; proc(y); y := x; end

x := 2;

call h(:g);

end; H1

{z = 2} / {h: proc...end}

(9) {true} call f( ) {z = 2}/

{h: proc...end, f: proc...end} H8

(10) {true} begin z := 3; call f( ); end/

{h: proc...end, f: proc...end} H5, H4a, b

(11) {true} begin

h: proc (:P);...end;

f: proc ( );...end;

```
        z := 3;                                   By two applications
                                                  of H2.
        call  f(  );
        end;
            {z = 3}/ϕ
```

This completes the proof. In the next section we will show that axioms H1 - H8 above are sound and, in a certain sense, complete.

5.4 Soundness and Completeness

Consider the programming language described in section 5.2 with the restriction that procedures be nonrecursive. (Thus we are allowing (a) static scope, (b) global variables, and (c) nonrecursive procedures with procedure parameters.)

Theorem: 5.4.1

For all L_A, L_E, and $\mathfrak{a}$ if (a) T is a complete proof system for L_A relative to $\mathfrak{a}$ and if (b) L_A is expressive relative to L_E and $\mathfrak{a}$, then for all partial correctness assertions of the form {P} A {Q}/D we have

$$\models_{\mathfrak{a}} \{P\}\ A\ \{Q\}/D \Leftrightarrow \vdash_{H,T} \{P\}\ A\ \{Q\}/D$$

Proof: The proof will be divided into two parts. In the first part we prove soundness i.e. if $\vdash_{H,T} \{P\}\ A\ \{Q\}/D$ then $\models_{\mathfrak{a}} \{P\}\ A\ \{Q\}/D$. In the second part we prove completeness, i.e. if $\models_{\mathfrak{a}} \{P\}\ A\ \{Q\}/D$ then $\vdash_{H,T} \{P\}\ A\ \{Q\}/D$.

Soundness: We must show that for each of the rules of inference H1-H9 that if all of the hypothesis of the rule are true (with respect to $\mathfrak{a}$) then the conclusion will be true also. Here we examine H1, H2, and

H9 -- the remaining ones are left to the reader.

First H1. Assume that $\{U \wedge x^i = e_0\}$ begin A $\frac{x^i}{x}$ end $\{V\}/D$ is true respect to $\mathfrak{a}$. Let π be the environment corresponding to D. Thus by the definition of partial correctness we have

$$\forall\ s, s'\ [s \in (U \wedge x^i = e^0) \wedge s' = M[\text{begin A } \frac{x^i}{x} \text{ end}]\ (s)(\pi) \Rightarrow s'\Big|_{free(V)} \in V]$$

Since x^i does not occur free in V, if we let $s'' = DEL(s', x^i)$ then $s'\Big|_{free(V)} \in V$ iff $s'\Big|_{free(V)} \in V$.

Thus

$$\forall\ s, s''\ [s \in (U \wedge x^i = e_0) \wedge s'' = DEL(M[\text{begin A } \frac{x^i}{x} \text{ end}]\ (s)(\pi), x^i) \Rightarrow s''\Big|_{free(V)} \in V]$$

Let $s^* = DEL(s, x^i)$. Then

$M[\text{begin new } x; A \text{ end}]\ (s^*)(\pi) = DEL(M[\text{begin A } \frac{x^i}{x} \text{ end}]\ (s)(\pi), x^i)$

Hence it follows that

$$\forall\ s^*, s''[s^* \in U \wedge s'' = M[\text{begin new } x; A \text{ end}](s^*)(\pi) \Rightarrow s''\Big|_{free(V)} \in V]$$

or $\models_{\mathfrak{a}} \{U\}$ begin new x; A end $\{V\}/D$.

Next we consider H2. Assume that

$\models_{\mathfrak{a}} \{U\}$ begin A $\frac{q^i}{q}$ end $\{V\}/D \cup \{q^i: \text{proc}(\bar{x}:\bar{p});\ K\ \frac{q^i}{q}\ \text{end}\}$

is true with respect to $\mathfrak{a}$. Let π be the environment corresponding to D and let $\pi' = \pi[q^i \leftarrow \langle(\bar{x}:\bar{p})\ K, \frac{q^i}{q}\rangle]$.

By the definition of partial correctness

$$\forall\ s, s'[s \in U \wedge s' = M[\text{begin A } \frac{q^i}{q} \text{ end}]\ (s)(\pi') \rightarrow s'\Big|_{free(V)} \in V]$$

But $M[\text{begin } q: \text{proc}(\bar{x}:\bar{p}); K \text{ end}; A \text{ end}]\ (s)(\pi) = M[\text{begin A } \frac{q^i}{q} \text{ end}]\ (s)(\pi')$

Thus

$$\forall\, s, s'[s \in U \wedge s' = M[\text{begin } q\text{: proc } (\bar{x}:\bar{p});\ K \text{ end};\ A \text{ end}]\,(s)(\pi) \Longrightarrow s'\Big|_{\text{free}(V)} \in V]$$

or

$\models_{\mathfrak{a}} \{U\}$ begin q: proc $(\bar{x}:\bar{p})$; K end; A end $\{V\}/D$ as required.

Finally we consider H9. Assume that $\{U\}\ K\ \frac{\bar{a}}{\bar{x}}\ \frac{\bar{P}}{\bar{p}}\ \{V\}/D$ which includes P is true in $\mathfrak{a}$. Let π be the environment map corresponding to D then we have by the definition of partial correctness that

$$\forall\ s, s'[s \in U \wedge s' = M[\ K\ \frac{\bar{a}}{\bar{x}}\ \frac{\bar{P}}{\bar{p}}\ ]\,(s)(\pi) \Rightarrow s'\Big|_{\text{free}(V)} \in V]$$

But $M[\text{call } q(\bar{a}:\bar{P})](s)(\pi') = M[K\ \frac{\bar{a}}{\bar{x}}\ \frac{\bar{P}}{\bar{p}}\ ]\,(s)(\pi')$

where $\pi' = \pi[q \leftarrow \langle(\bar{x}:\bar{p}),\ K\ \rangle]$. Since recursion has been disallowed, we have

$$M[K\ \frac{\bar{a}}{\bar{x}}\ \frac{\bar{P}}{\bar{p}}\ ]\,(s)(\pi') = M[K\ \frac{\bar{a}}{\bar{x}}\ \frac{\bar{P}}{\bar{p}}\ ]\,(s)(\pi)$$

thus

$$M[\text{call } q\ (\bar{a}:\bar{p})]\,(s)(\pi') = M[K\ \frac{\bar{a}}{\bar{x}}\ \frac{\bar{P}}{\bar{p}}\ ]\,(s)\,(\pi)$$

It follows that

$$\forall\ s, s'[s \in U \wedge s' = M[\text{call } q\ (\bar{a}:\bar{P})]\,(s)(\pi') \Rightarrow s'\Big|_{\text{free}(V)} \in V]$$

or that $\models_{\mathfrak{a}} \{U\}$ call q $(\bar{a}:\bar{P})\{V\}/D \cup \{q$: proc $(\bar{x}:\bar{p})$; K end$\}$ which is the desired conclusion. The other cases H3-H8 are similar to the ones considered above.

Completeness: We show that if T is a complete proof system for L_A

relative to $\mathfrak{A}$ and if L_A is expressive relative to L_E and $\mathfrak{A}$, then for all partial correctness assertions of the form $\{U\}\ A\ \{V\}/D$.

$$\models_{\mathfrak{A}} \{U\}\ A\ \{V\}/D \Longrightarrow \vdash_{H,T} \{U\}\ A\ \{V\}/D$$

The proof will be by induction on the structure of A. We will show that $\vdash_{H,T} \{U\}\ A\ \{V\}/D$ if A is an atomic statement of the programming language (e.g. an assignment statement). We will also show that if A is a composite statement (e.g. "begin new x; B end" or "while b do A") then to establish $\{U\}\ A\ \{V\}/D$ it is sufficient to first establish $\{U'\}\ A'\ \{V'\}/D$ where A' is either shorter than A or involves fewer procedure calls.

Case(1): Suppose that A is "begin new x; B end" and that $\{U\}$ begin new x; B end $\{V\}/D$ is true with respect to $\mathfrak{A}$.

Then by the definition of partial correctness

$$\forall\ s, s'[s \in U \wedge s' = M[\text{begin new } s;\ B \text{ end}]\ (s)(\pi) \Rightarrow s'\big|_{free(V)} \in V]$$

where π is the environment corresponding to D. But

$$M[\text{begin new } x;\ B \text{ end}]\ (s)(\pi) = DEL(M[\text{begin } B\ \tfrac{x^i}{x} \text{ end}]\ (s^*)(\pi),\ x^i)$$

where i is the index of the first program identifier not appearing in A, π, or $DOM(s)$ and $s^* = s[x^i \leftarrow e_0]$. Since x^i does not occur free in U or V, we have

(a) if $s^* = s[x^i \leftarrow e_0]$ then $s \in U$ iff $s^* \in U \wedge x^i = e_0$, and

(b) if $s' = DEL(s'', x^i)$ then $s'\big|_{free(V)} \in V$ iff $s''\big|_{free(V)} \in V$.

It follows that if i is the index of the first program identifier not appearing free in A, D, or $free(U)$ then

$$\forall\ s^*, s''[s^* \in (U \wedge x^i = e_0) \wedge s'' = M[\text{begin } B\ \tfrac{x^i}{x} \text{ end}](s^*)(\pi) \Longrightarrow s''\big|_{free(V)} \in V]$$

Applying the definition of partial correctness again we have that $\{U \wedge x^i = e_0\}$ begin $B\frac{x^i}{x}$ end $\{V\}/D$.

Since "begin $B\frac{x^i}{x}$ end" is strictly shorter than "begin new x; B end", we have by the induction hypothesis that $\vdash_{H,T} \{U \wedge x^i = e_0\}$ begin $B\frac{x^i}{x}$ end $\{V\}/D$.

Using the fact that i is the index of the first program identifier not appearing in A, D, or U, we may use H1 to conclude $\vdash_{H,T}$ $\{U\}$ A $\{V\}/D$ as required.

Case (2): Suppose that A is "begin q: proc $(\bar{x}:\bar{p})$; K end; B end" and that $\{U\}$ begin q: proc $(\bar{x}:\bar{p})$; K end; B end $\{V\}/D$ is true with respect to $\mathfrak{a}$. By the definition of partial correctness

$$\forall\ s,s'[s \in U \wedge s' = M[\text{begin } q\text{: proc } (\bar{x}:\bar{p}); \text{ K end; B end}](s)(\pi) \Longrightarrow s'\Big|_{free(V)} \in V]$$

where π is the environment corresponding to D. But M[begin q: proc $(\bar{x}:\bar{p})$; K end; B end](s)(π) = M[begin $B\frac{q^i}{q}$ end] (s)(π') where i is the index of the first procedure identifier not occurring in A or π and $\pi' = \pi[q^i \leftarrow \langle(\bar{x}:\bar{p}), K\frac{q^i}{q}\rangle]$

thus

$$\forall\ s, s'[s \in U \wedge s' = M[\text{begin } B\frac{q^i}{q} \text{ end}] (s)(\pi') \Longrightarrow s'\Big|_{free(V)} \in V]$$

If we let D' be D U $\{q^i$: proc $(\bar{x}:\bar{p})$; K $\frac{q^i}{q}$ end$\}$ then π' is the environment corresponding to D' and by another application of the definition of partial correctness, $\models_{\mathfrak{a}} \{U\}$ begin $B\frac{q^i}{q}$ end $\{V\}/D'$.

Since "begin $B\frac{q^i}{q}$ end" is strictly shorter than "begin q: proc $(\bar{x}:\bar{p})$; K end; B end" we conclude that $\vdash_{H,T} \{U\}$ begin $B\frac{q^i}{q}$ end $\{V\}/D$.

It follows by axiom H2 that $\vdash_{H,T} \{U\}$ A $\{V\}/D$ also.

Case (3): Suppose that A is "b*A_1" and that $\{U\}$ b*A_1 $\{V\}/D$ is

true with respect to $\mathfrak{a}$. Since the assertion language L_A is expressive with respect to L_E and $\mathfrak{a}$, we know that there is a formula in L_A which represents $R[b*A_1](\pi)(V)$ where π is the environment corresponding to D. The reader may verify that each of the following assertions is correct.

(i) $\vdash_{\mathfrak{a}} U \rightarrow R[b*A_1](\pi)(V)$ and hence $\vdash_T U \rightarrow R[b*A_1](\pi)(V)$ since T is assumed to be a complete proof system for L_A.

(ii) $\vdash_{\mathfrak{a}} \{R[b*A_1](\pi)(V) \wedge b\}\ A_1\ \{R[b*A_1](\pi)(V)\}/D$ -- so $\vdash_{H,T} \{R[b*A_1](\pi)(V) \wedge b\}\ A_1\ \{R[b*A_1](\pi)(V)\}/D$ follows by the induction hypothesis since A_1 is shorter than $b*A_1$.

(iii) $\vdash_{\mathfrak{a}} R[b*A_1](\pi)(V) \wedge \sim b \rightarrow V$ and hence $\vdash_{H,T} R[b*A_1](\pi)(V) \wedge \sim b \rightarrow V$ since T is a complete proof system for L_A.

The desired result $\vdash_{H,T} \{U\}\ A\ \{V\}/D$ follows by and application of the while axiom and the rule of consequence.

Case(4): Suppose that A is "call $q(\bar{a}:\bar{P})$", that $\{U\}$ call $q(\bar{a}:\bar{P})$ $\{V\}/D$ is true with respect to $\mathfrak{a}$, that π is the environment corresponding to D, and that $\pi(q) = \langle(\bar{x}:\bar{p}),\ K\rangle$. By the definition of partial correctness:

$$\forall\ s, s'[s \in U \wedge s' = M[\text{call } q(\bar{a}:\bar{P})](s)(\pi) \Longrightarrow s'\Big|_{\text{free}(V)} \in V]$$

But $M[\text{call } q(\bar{a}:\bar{P})](s)(\pi) = M[K\ \frac{\bar{a}}{\bar{x}}\ \frac{\bar{P}}{\bar{p}}\ ](s)(\pi)$

Hence

$$\forall\ s, s'[s \in U \wedge s' = M[K\ \frac{\bar{a}}{\bar{x}}\ \frac{\bar{P}}{\bar{p}}\ ](s)(\pi) \Longrightarrow s'\Big|_{\text{free}(V)} \in V]$$

and we have

$\vdash_{\mathfrak{a}} \{U\}\ K\ \frac{\bar{a}}{\bar{x}}\ \frac{\bar{P}}{\bar{p}}\ \{V\}/D$. Since recursion is not allowed, $K\ \frac{\bar{a}}{\bar{x}}\ \frac{\bar{P}}{\bar{p}}$ does not involve the procedure q. Thus by the inductive assumption

$\vdash_{H,T} \{U\}\ K\ \frac{\bar{a}}{\bar{x}}\ \frac{\bar{P}}{\bar{p}}\ \{V\}/D$.

By axiom H9 we get

$\vdash_{H,T}$ {P} call q($\bar{a}$:$\bar{P}$) {Q}/D $\cup$ {q: proc($\bar{x}$:$\bar{p}$); K end}

Since "Q: proc(x:p); K end" $\in$ D, we get the desired result that

$\vdash_{H,T}${P} call q($\bar{a}$:$\bar{P}$){Q}/D.

Cases 5-8 correspond to A being one of

a) "begin B; B_1^* end"

b) "begin end"

c) "x := e"

d) "b $\rightarrow$ A_1, A_2

Since these cases are handled in a manner similar to that used in cases 1-4, they will be left to the interested reader. This completes the proof of theorem 1.

If we disallow procedure parameters, then it is possible to obtain a complete set of Hoare-like axioms even if the procedures may be recursive. The axioms given in Gorelick [G075] can be used almost verbatim in spite of the static scope requirement. We replace H9 by axioms R1 - R5 below.

R1) $\dfrac{\{P\}\ \text{call}\ r\ (\bar{x})\ \{Q\}/D \vdash \{P\}\ K(r)\ \{Q\}/D,\quad r \text{ a dummy procedure name}}{\{P\}\ \text{call}\ q(\bar{x})\ \{Q\}/D \cup \{q\text{: proc}(\bar{x});\ K(q);\ \text{end}\}}$

provided that D does not contain a procedure "q: proc($\bar{x}$); K' end" which is different from "q: proc($\bar{x}$); K(q) end".

5.4.2 Definition:

Let the procedure q have declaration "q: proc($\bar{x}$); K end."

A variable y is <u>active</u> with respect to "call $q(\bar{a})$" if y is either global to K or y is in $\bar{a}$ (i.e. is an actual parameter of the call). If y is not active with respect to "call $q(\bar{a})$", then y is said to be <u>inactive</u> (with respect to that particular call). Similarly, a term of the assertion language L_A is <u>inactive</u> if it contains only inactive variables. A substitution σ is <u>admissible</u> with respect to "call $q(\bar{a})$" provided that it is a substitution of inactive terms for inactive variables.

R2) $\dfrac{\{P\}\ \text{call}\ q(\bar{a})\ \{Q\}/D}{\{P\sigma\}\ \text{call}\ q(\bar{a})\ \{Q\sigma\}/D}$ provided σ is admissible with respect to "call $q(\bar{a})$"

R3) $\dfrac{\{P(u_0)\}\ \text{call}\ q(\bar{a})\{Q(u_0)\}/D}{\{\exists u_0 P(u_0)\}\ \text{call}\ q(\bar{a})\ \{\exists\, u_0 Q(u_0)\}/D}$ provided that u_0 is inactive in "call $q(\bar{a})$"

(Rule R3 above is not absolutely necessary but its inclusion simplifies the proof of completeness.)

R4) $\dfrac{\{P\}\ \text{call}\ q(a)\ \{Q\}/D}{\{P \wedge T\}\ \text{call}\ q(a)\ \{Q \wedge T\}/D}$ provided that no variables which occur free in T are active in "call $q(\bar{a})$".

R5) $\dfrac{\{P\}\ \text{call}\ q(\bar{x})\ \{Q\}/D}{\{P\frac{\bar{a}}{\bar{x}}\}\ \text{call}\ q(\bar{a})\ \{Q\frac{\bar{a}}{\bar{x}}\}/D}$ provided that no variable free in P or Q occurs in $\bar{a}$ but not in in the corresponding position of $\bar{x}$. ($\bar{x}$ is the list of <u>formal parameters</u> of q).

<u>Theorem 5.4.3:</u>

Let H' be the set of axioms H1-H8 together with axioms R1 - R5. Restrict the programming language of section 5.2 so that procedure parameters are disallowed. For all L_A, L_E and $\mathfrak{A}$, if (a) T is a complete proof system for L_A relative to $\mathfrak{A}$ and (b) L_A is expressive relative to L_E and $\mathfrak{A}$, then for all partial correctness assertions of

the form

$\{P\}\ A\ \{Q\}/D$, we have

$\models_{\mathfrak{a}} \{P\}\ A\ \{Q\}/D \Longleftrightarrow \vdash_{H,T} \{P\}\ A\ \{Q\}/D$

The proof of theorem 5.2 involves modifying the argument of [G075] to apply to static scope. Since static scope may be handled by the techniques described earlier in this section, the proof will be omitted.

5.5 Recursive Procedures with Procedure Parameters

In this section we prove that there cannot be a sound, complete set of Hoare-like axioms for a programming language with (a) global variables, (b) static scope, and (c) recursive procedures with procedure parameters (sound and complete in the sense of section 3. The effect of stronger notion of expressibility, etc. will be discussed later.) To make the theorem stronger, we disallow calls of the form Call P(...P) i.e. we require that actual procedure parameters be names of procedures with no procedure formal parameters. Calls of the form Call P(...,P) appear to be necessary in order to directly simulate the lambda calculus by parameter passing.

Main Lemma 5.5.1:

Consider a programming language allowing recursive procedures with procedure parameters (assuming static scope and allowing global variables) then the halting problem for such a language is undecidable even under a finite interpretation $\mathfrak{a}$.

The proof of the lemma uses techniques developed by Jones and Muchnick [J075]. Before we outline the proof note that the lemma is of true for flowchart schemes or while schemes since in each of these cases if $|\mathfrak{a}| < \infty$ the program may be viewed as a finite state machine, and we may test for termination (at least theoretically) by watching the execution

sequence of the program to see if any program state is repeated. In the case of recursion one might expect that the program could be viewed as a type of push down automaton (for which the halting problem is decidable). This is not the case if we allow procedures as parameters. The Algol 60 execution rule, which says that procedure calls are elaborated in the environment of the procedure's declaration rather than in the environment of the procedure call, allows the program to access values normally buried in the runtime stack without first "popping the top" of the stack.

As in Jones and Muchnick [J075] we show that it is possible to simulate a queue machine which has three types of instruction a) Enqueue x -- add the value of x to the rear of the queue, b) Dequeue x -- remove the front entry from the queue and place in x, and c) If x = y then L_1 else L_2 -- conditional branch.
(By using procedures with procedure parameters instead of call by name as in Jones and Muchnick, we are able to avoid introducing any non Algol 60 constructs.)

Suppose that the Queue machine program to be simulated is given by

$$P = 1 : I_1; \ldots K : I_k$$

then the simulation program (in the language of section 5.2) has the form

```
Begin
    new I, Dummy, X1,...Xn;
    diverge: proc; while true do null end: end diverge;
    sim: proc(I:back);
        begin new top;
            up: proc(end_of_queue, next_to_last:);
                begin
```

```
                    If  top < 0
                    then begin
                         end_of_queue := top;
                         next_to_last := 1;
                         end;
                    else begin
                         call back(end_of_queue, next_to
                            last:);
                         If next_to_last=1
                         then begin
                              top := -top;
                              next_to_last=0;
                         end;
                    end;
               end up;
          If I=1 then "I1" else
          If I=2 then "I2" else
               .
               .
               .
          If I=K then "IK" else null;
     end;
end sim;
I:=1;
call sim(I : diverge);
end;
```

where "I_j" is described by the three cases below.

(A) If I_j is "j:enqueue A;" then replace by:

```
begin
     If I=1 then top := -A; else top := A;
     I := I+1;
     Call sim(I: up);
```

```
            return;
        end;
```

(B) If I_j is "j:dequeue x" then replace by

```
begin

    call back (x: dummy);

    x := -x;

    I := I+1;

end;
```

(C) If I_j is "If $x_p = x_m$ then go to n" then replace by

```
begin

    If x_p = x_m

    then I := n else I := I+1;

end;
```

Note that I must have range of values $1 \leq I \leq K+1$. If $|\mathfrak{a}| < K + 1$ then it is necessary to represent I by $I_1, I_2 \ldots I_d$ when $I_1, I_2, \ldots I_d$ is the binary representation of I. This also eliminates the need for the arithmetic operation I = I+1. The variables $x_1 \ldots x_n$ represent the program variables of the program P which is being simulated.

Now we return to the proof of the main theorem of this section. Suppose that we had a sound, complete Hoare-like proof system H for programs of the type described at the beginning of this section. Then for all L_A, L_E, and $\mathfrak{A}$, if

1) T is a complete proof system for L_A, and $\mathfrak{A}$ and

2) L_A is expressive relative to L_E and $\mathfrak{A}$, then we should have

$$\vDash_{\mathfrak{a}} \{P\}\ A\ \{Q\}/_{\phi} \Longrightarrow \vdash_{H,T} \{P\}\ A\ \{Q\}/_{\phi}$$

We show that this leads to a contradiction. Choose $\mathfrak{a}$ to be a finite interpretation with $|\mathfrak{a}| \geq 2$.

First observe that T may be chosen in a particularly simple manner; in fact, there is a decision procedure for truth of formulas in L_A relative to $\mathfrak{a}$. Secondly note that L_A is expressive relative to L_E and $\mathfrak{a}$. This was shown by proposition 5.2.4 of section 2 since $\mathfrak{a}$ is finite. Thus both hypothesis 1) and 2) are satisfied. From the definition of partial correctness, we see that {true} A {false}/ϕ iff A diverges for the initial values of its global variables. By the main lemma above, we conclude that the set of program A such that $\vdash_{\mathfrak{a}}$ {true} A {false} /ϕ holds is not r.e. On the other hand if we had

$$\vdash_{\mathfrak{a}} \{\text{true}\}\ A\ \{\text{false}\}/\phi \Longleftrightarrow \vdash_{H,T} \{\text{true}\}\ A\ \{\text{false}\}\ /\phi$$

Then we could enumerate those programs A such that $\vdash_{\mathfrak{a}}$ {true} A {false}/ϕ holds (simply enumerate all possible proofs and use the decision procedure for T to check applications of the rule of consequence). This, however, is a contradiction.

5.6 Coroutines

A coroutine will have the form:

coroutine; Q_1; Q_2; end

Q_1 is the main-routine; execution begins in Q_1 and also terminates in Q_1 (the requirement that execution terminate in Q_1 is not necessary but simplifies the axiom for coroutines). Otherwise Q_1 and Q_2 behave in identical manners. If an "exit" statement is encountered in Q_1, the next statement to be executed will be the statement following the last "resume" statement executed in Q_2. Similarly, the execution of a "resume" statement in Q_2 causes execution to be restarted following the last exit statement executed in Q_1. If the "exit" ("resume") statement occurs within

a call on a recursive procedure then execution must be restarted in the correct activation of the procedure. A simple example of coroutines is given below:

```
coroutine;
    begin
        while x-y < z do
            x := x+2
            y := y+2
            exit;
            end;
        end;
    begin
        while true do
            y := y-1;
            resume;
            y := y-2;
            resume;
            end;
        end;
    end;
```

Note that if x and y are zero initially, then when the coroutine terminates $y = \lceil z/3 \rceil$.

5.7 Semantics of Coroutines

M[A] (s)(π) is defined as before except for M[coroutine 5.7 Q_1; Q_2 end] (s)(π) which is defined in terms cf two mutually recursive procedures C1 and C2 (one for each coroutines) as follows:

M[coroutine Q_1; Q_2 end] (s)(π) = Cl[Q_1, π, Q_2, π, s] where Cl[R_1, π_1, R_2, π_2, s] is defined by cases on R_1

(1) begin new x; A end; $R_1' \rightarrow$ Cl[begin A $\frac{x^i}{x}$ end; R_1', π_1, R_2, π_2, s'] where i is the index of the first unused program identifier and $s' = s[x^i \leftarrow e_0]$.

(2) begin q: proc($\bar{x}$); K end A end; $R_1' \rightarrow$ Cl[begin A $\frac{q^i}{q}$ end; R_1', π_1', R_2, π_2, s] where i is the index of the first unused procedure name and $\pi_1' = \pi_1[q^i \leftarrow \langle(\bar{x}), K\rangle]$

(3) begin A end; $R_1' \rightarrow$ Cl[A; R_1', π_1, R_2, π_2, s]

(4) exit; $R_1' \rightarrow$ C2[R_1', π_1, R_2, π_2, s]

(5)-(8) cases corresponding to assignment, conditional, while, and procedure call -- see section 2.

(9) Λ (i.e. R_1 is the empty string) $\rightarrow$ s

The definition of C2[R_1, π_1, R_2, π_2, s] is the dual of the definition given above except that C2[R_1, π_1, Λ, π_2, s] = Cl[R_1, π_1, Λ, π_2, s]. Thus execution of the coroutine always terminates in Q_1. Note also that the semantics given above do not allow for nested coroutines. The semantics could be modified to handle this case, but nesting of coroutines is unnecessary in order to illustrate the problem that we are interested in here.

5.8 Axioms for Coroutines (no recursion)

In this section we give a "good" set of axioms for coroutines (nonrecursive procedures only) and describe a technique for proving correctness of coroutines which is based on the addition of "auxilliary variables". This technique was suggested in part S. Owicki. It is different from the technqiue described by Clint [CL73] in that the auxilliary variables represent program counters

(therefore have bounded magnitude) rather than arbitrary stacks.

Axioms for Coroutines:

C1. $\{P'\}$ exit $\{R'\} \vdash \{P \wedge b\}\ Q_1\ \{R\}$

$$\frac{\{R'\} \text{ resume } \{P'\} \vdash \{P' \wedge b\}\ Q_2\ \{R'\}}{\{P \wedge b\} \text{ coroutine } Q_1;\ Q_2 \text{ end } \{R\}}$$

provided no variable free in b is global to Q_1.

C2. $$\frac{\{P\} \text{ exit } \{Q\}}{\{P \wedge C\} \text{ exit } \{Q \wedge C\}}$$

provided that C does not contain any free variables that are changed by Q_2. (Here we assume that "exit" occurs in statement Q_1 of "coroutine Q_1; Q_2 end")

C3. $$\frac{\{P\} \text{ resume } \{Q\}}{\{P \wedge C\} \text{ resume } \{Q \wedge C\}}$$

provided that C does not contain any free variables that are changed by Q_1. (Here we assume that "resume" occurs in statement Q_2 of "coroutine Q_1; Q_2 end").

C4. Let AV be a set of variables such that $x \in AV \Longrightarrow x$ appears in S' only in assignment $y := E$, with $y \in AV$. Then if P and Q are assertions which do not contain free any variables from AV and if S is obtained from S' by deleting all assignments to variables in AV, then

$$\frac{\{P\}\ S'\ \{Q\}}{\{P\}\ S\ \{Q\}}$$

(This axiom was taken from [OW76].)

We illustrate the above axioms with an example. We show that $\{x=0 \wedge y=0\}\ R\ \{y = \lceil z/3 \rceil\}$ where R is the coroutine given at the beginning of section 7. Our strategy in carrying out the proof will be to introduce auxilliary variables which distinguish the various "exit" and "resume" statements from each other and then use axiom C4 to delete these unnecessary variables as the last step of the proof.

Thus we will actually prove

```
{x=0 ∧ y=0}
        i := 0;
        j := 0;
        coroutine;
             Begin
                  while x-y < z do
                       x := x+2;
                       y := y+2;
                       i := 1;
                       exit;
                       end;
                  end;
             Begin
                  while true do
                       y := y-1;
                       j := 1;
                       resume;
                       y := y-2;
                       j := 2;
                       resume;
                       end;
```

end;

{y = ⌈z/3⌉}

Choose P = {x=0 ∧ y=0 ∧ i=0 ∧ j=0}, b = {j=0}

R = {y = ⌈z/3⌉ }

P' = {(x=2 ∧ y=2 ∧ j=0) ∨ (x=4y-8 ∧ j=1) ∨ (x=4y-6 ∧ j=2)}

R' = {(x=4y-2 ∧ j=1) ∨ (x=4y ∧ j=2)}

The invariant for the while loop of the first routine is

I_1 = {(x=0 ∧ y=0 ∧ j=0) ∨(x=4y-2 ∧x-y ≤ z ∧ j=1) ∨
(x=4y ∧ x-y ≤ z+1 ∧ j=2)}.

The invariant for the while loop of the second routine is

I_2 = {(x=2 ∧ y=2 ∧ j=2) ∨ (x=4y-6 ∧ j=2)}. It is easily checked using axioms C2-C4 together with the axioms for the assignment statement and the while statement that

$$\{P'\}\text{exit}\{R'\} \vdash \{P\}\ Q_1\ \{R\}$$

and

$$\{R'\}\ \text{resume}\ \{P'\} \vdash \{P' \wedge b\}\ Q_2\ \{R'\}$$

The desired conclusion then follows by Axiom C1.

The technqiue of adding auxilliary variables is easily formalized (the pattern should be clear from the above example). Thus, in general, we are able to prove:

Theorem 5.8.1

Consider the language described in sections 5.6 and 5.7 including the coroutine statement but requiring that procedures be non-recursive. Let H" be the Hoare-like axiom system consisting of axioms H1-H9 together with C1-C4. If T, L_A, L_E, and $\mathfrak{A}$ satisfy the conditions

A) T is a complete proof system for L_A and $\mathfrak{a}$ and

B) L_A is expressive relative to L_E and $\mathfrak{a}$, then

$\models_{\mathfrak{a}} \{P\}\ A\ \{Q\} \Rightarrow \vdash_{H,T} \{P\}\ A\ \{Q\}$.

5.9 Coroutines and Recursion:

We show that it is impossible to obtain a sound-complete system of Hoare-like axioms for a programming language allowing both coroutines and recursion provided that we do not assume a stronger type of expressibility than that defined in section 2. (We will argue in section 10 that the notion of expressibility introduced in section 2 is the natural one. We will examine the consequences of adopting a stronger definition of expressibility.)

Lemma 5.9.1.

Consider the programming language described in section 7 with coroutines and recursive procedures, then the halting problem for programs in such a language is undecidable even under a finite interpretation $\mathfrak{a}$.

The proof of the lemma is similar to the proof of the main lemma of section 5.5; however this time we reduce to the halting problem for a two stack machine rather than a queue machine. The simulation program will be a coroutine with one of its component routines controlling each of the two stacks. Each stack is represented by the successive activations of a recursive procedure local to one of the routines. Thus, stack entries are maintained by a variable local to the recursive procedure, deletion from a stack is equivalent to a procedure return, and additions to a stack are accomplished by recursive calls of the procedure. The simulation routine is given in outline form below:

```
Prog_counter := 1;
Coroutine
     begin
          stack_1: proc(empty);
               new top, progress;
               progress := 1;
               while progress=1 do
                    if prog_counter = 1 then "I1" else
                    if prog_counter = 2 then "I2" else
                    .
                    .
                    .
                    if prog_counter = K then "Ik" else NULL;
                    end;
               end stack_1;
          call stack_1 (1);
     end;
     begin
          stack_2: proc (empty);
               new top, progress;
               progress := 1;
               while progress=1 do
                    if prog_counter = 1 then "I1*" else
                    if prog_counter = 2 then "I2*" else
                    .
                    .
                    .
                    if prog_counter = K then "IK*" else null;
                    end;
               end stack_2;
          call stack_2 (1);
     end;
end;
```

where "I_1,..."I_K","I_1^*",..."I_K^*" are encodings of the program for the two

stack machine being simulated. Thus, for example, in the procedure STACK_1 we have the following cases:

(1) if I_j is PUSH X ON STACK_1, "I_j" will be

```
begin
     top = x;
     prog_counter := prog_counter + 1;
     call stack_(0);
end;
```

(2) if I_j is POP X FROM STACK_1, "I_j" will be

```
begin
     if empty = 1 then null;
     else begin
          prog_counter=prog_counter +1;
          x = top;
          progress := 0;
          end;
     end;
```

(3) if I_j is PUSH X ON STACK_2 or POP X FROM STACK_2 "I_j" will simply be

```
begin
     exit;
end;
```

A similar encoding $I_1^*,\ldots I_K^*$ for the copy of the program within procedure stack_2 may be given. Statements of the form "prog_counter := prog_counter+1" may be eliminated by introducing a fixed number of new variables to represent the binary representation of "prog-counter".

The remainder of the proof that it is impossible to obtain a sound-complete set of Hoare-like axioms for coroutines with recursive procedures is almost identical to the proof given in section 4 for recursive procedures with procedure parameters and will be omitted.

5.10 Discussion of Results and Open Problems:

A number of open problems are suggested by the above results. First we consider the problem of obtaining Hoare-like axioms for recursive procedures with procedure parameters. The proof that no good set of axioms can exist in the general case depends heavily on the fact that procedures can access global variables--if global variables are disallowed or made read-only is it possible to obtain a complete set of axioms? Similarly, if we changed from static scope to dynamic scope or allowed only the names of external procedures to occur as actual procedure parameters, would the problem of proving correctness be simplified?

In the case of coroutines and recursion the most important question seems to be whether a stronger form of expressibility might give completeness. The result of section 5.8, seems to require that any such notion of expressibility be powerful enough to allow assertions about the status of the run-time stack(s). Clint [CL73] suggests the use of stack-valued auxilliary variables to prove properties of coroutines which involve recursion. It seems likely that a notion of expressibility which allowed such variables appears counter to the spirit of the axiomatic correctness. If a proof of a recursive program can involve the use of stack valued variables, why not simply replace the recursive procedures themselves by stack operations. The purpose of recursion in programming languages is to free the programmer from the details of implementing recursive constructs via stacks. Further evidence for the naturalness of the notion of expressibility that we have

used is the fact that with either a) coroutines and non-recursive procedures or b) recursive procedures and no coroutines, we can obtain sound, complete sets of axioms under this notion of expressibility.

We note that the main application of procedures with procedure parameters is in numerical analysis (where on might wish to make the integrand a parameter of an integration procedure.) Here, however, procedures are rarely recursive and hence can be handled by the techniques of section 5.3. Similarly coroutines are most frequently used in I/O routines which do not involve recursion and which can be handled by the methods of section 5.7(if appropriate I/O axioms are introduced).

Finally we note that the technique of section 5.4 and 5.8 may be applied to a number of other programming language features including a) call by name with functions and global variables, b) pointer variables with retention, c) pointer variables with recursion, and d) label variables with retention. All of these features appear to be inherently difficult to prove correct.

REFERENCES

[CL72] Clint, M. and C. A. R. Hoare. Program Proving: Jumps and Functions. Acta Informatica, Vol. 1, pp. 214-224, 1972.

[CL73] Clint, M. Program Proving: Coroutines. Acta Informatica, Vol. 2, pp. 50-63, 1973.

[CO75] Cook, S. A. Axiomatic and Interpretative Semantics for an Algol Fragment. Technical Report 79, Department of Computer Science, University of Toronto, 1975 (to be published in SCICOMP).

[DE73] deBakker, J. W. and L. G. L. Th. Meertens. On the Completeness of the Inductive Assertion Method. Mathematical Centre, December 1973.

[DE75] deBakker, J. E. Fixed Point Semantics and Dijkstra's Fundamental Invariance Theorem. Mathematical Centre, January 1975.

[DE75A] deBakker, J. W. Flow of Control in the Proof Theory of Structured Programming. Mathematical Centre and Free University, 1975.

[DI73] Dijkstra, E. E. A Simple Axiomatic Basis for Programming Language Constructs. Lecture notes from the International Summer School on Structured Programming and Programmed Structures, Munich, Germany, 1973.

[DO74] Donahue, James. Mathematical Semantics as a Complementary Definition for Axiomatically Defined Programming Language Constructs, in Donahue, et al., Three Approaches to Reliable Software: Language Design, Dyadic Specification, Complementary Semantics. Technical Report CSRG-45, Computer Systems Research Group, University of Toronto, December, 1974.

[FL67] Floyd, R. W. Assigning Meaning to Programs in Schwartz, J. T., ed. Mathematical Aspects of Computer Science Proc. Symposia in Applied Mathematics 19, pp. 19-32, Amer. Math. Soc., 1967.

[FO75] Fokkinga, M. C. Inductive Assertion Patterns for Recursive Procedures. Techn. University Delft Report, 1973.

[GO75] Gorelick, G. A Complete Axiomatic System for Proving Assertions about Recursive and Non-recursive Programs. Technical Report No. 75, Department of Computer Science, University of Toronto, January 1975.

[HO69] Hoare, C. A. R. An Axiomatic Approach to Computer Programming. CACM 12, 10 (October 1969), pp. 322-329.

[HO71] Hoare, C. A. R. Procedures and Parameters: An Axiomatic Approach. Symposium on Semantics of Algorithmic Languages, E. Engeler, Ed., Springer-Verlag, Berlin, pp. 102-116, 1971.

[HO74] Hoare, C. A. R. and P. E. Lauer. Consistent and Compementary Formal Theories of the Semantics of Programming Languages. Acta Informatica, Vol. 3, pp. 135-154, 1974.

[JO74] Jones, N. D. and S. S. Muchnick. Even Simple Programs Are Hard To Analyze. TR-74-6, Department of Computer Science, The University of Kansas, November, 1974, (to be published in JACM).

[MA70] Manna, Z. and A. Pnuefi. Formalization of Properties of Functional Programs. JACM 17, No. 3, pp. 555-569, 1970.

[MC73] McGowan, Clement and Jayadeo Misra. A Mathematical Basis for Dijkstra-Hoare Semantics. Technical Report No. 73-73, Center for Computer and Information Sciences, Brown University, November, 1973.

[MI73] Milner, R. Models of L. C. F. AIM-186/CS-332. Computer Science Department, Stanford University, 1973.

[OW76] Owicki, S. A Consistent and Complete Deductive System for the Verification of Parallel Programs. 8th Annual Symposium on Theory of Computing, 1976.

[SC70] Scott, D. Outline of a Mathematical Theory of Computation. Proceedings of Fourth Annual Princeton Conference on Information Science and Systems. Princeton, pp. 169-176, 1970.

[SC71] Scott, D. The Lattice of Flow Diagrams. E. Engeler (ed.), Semantics of Algorithmic Languages, Springer Notes in Mathematics, Vol. 188, pp. 311-366, 1971.

[WA76] Wand, M. A New Incompleteness Result for Hoare's System. 8th Annual Symposium on Theory of Computing, 1976.