A COMPLETE AXIOMATIC SYSTEM FOR PROVING DEDUCTIONS
ABOUT RECURSIVE PROGRAMS

DAVID HAREL
AMIR PNUELI
JONATHAN STAVI

FEBRUARY 1978
A COMPLETE AXIOMATIC SYSTEM
FOR PROVING DEDUCTIONS
ABOUT RECURSIVE PROGRAMS

David Harel
Amir Pnueli
Jonathan Stavi

February 1978
A COMPLETE AXIOMATIC SYSTEM FOR PROVING DEDUCTIONS ABOUT RECURSIVE PROGRAMS

David Harel
Amir Pnueli
Jonathan Stavi

February 1978

This research was supported by the National Science Foundation under contract no. MCS76-18461.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LABORATORY FOR COMPUTER SCIENCE

CAMBRIDGE MASSACHUSETTS 02139
Key Words:  Hoare's system
Partial correctness
Recursive programs
Relative completeness
Total correctness
A COMPLETE AXIOMATIC SYSTEM FOR PROVING DEDUCTIONS ABOUT
RECURSIVE PROGRAMS

David Harel
Massachusetts Institute of Technology
Cambridge, MA 02139

Amir Pnueli
Tel-Aviv University, Tel-Aviv, Israel

and
Jonathan Stavi
Bar-Ilan University, Ramat-Gan, Israel

2/21/77, revised 4/13/77

Abstract.

Denoting a version of Hoare's system for proving partial
correctness of recursive programs by H, we present an extension
D which may be thought of as H U (\forall, \forall, \exists, \forall) U H^{-1},
including the rules of H, four special purpose rules and inverse
rules to those of Hoare. D is shown to be a complete system (in
Cook's sense) for proving deductions of the form
\sigma_1, \ldots, \sigma_n \vdash \sigma
over a language, the wff's
of which are assertions in some assertion language L and partial
correctness specifications of the form p \land q. All valid formulae of L are
taken as axioms of D. It is shown that D is sufficient for
proving partial correctness, total correctness and program equivalence as
well as other important properties of programs, the proofs of which are
impossible in H. The entire presentation is worked out in the
framework of nondeterministic programs employing iteration and mutually
recursive procedures.

1. Introduction.

The axiomatic method of specifying semantics of programs, as given
by Hoare ([10], [11] and also [12]) lends itself very successfully to a
specific goal, namely that of proving partial correctness of specific
programs. A convenient description of the method employs an assertion
language L and a formal proof system H having as axioms all logically
valid formulae of L. A proof of a partial correctness specification
\sigma: p \land q where p,q are wff's in L, is carried out in H by
composing \sigma from more primitive program segments, starting from a finite
number of assumptions in L. A well known result is that the conventional
Hoare system and its variants are complete if L is strong enough to express all needed assertions. Various definitions of this strength are expressiveness of L (Cook[3]), or tidiness of all programs (Pratt[15]). Cook[3] showed that first order arithmetic is expressive, thus proving completeness of H for this important special case of L. Extensions of Hoare's system to cover recursion and mutual recursion have also been proved complete under similar conditions (see Gorelick[7], Harel et al[9]).

A suitable such system H can in fact be thought of as a formal system for proving the correctness of deductions of the form $\sigma_1, \ldots, \sigma_n \vdash p\{a\}q$ under the restriction that each of the $\sigma_i$ is a procedure declaration or a formula of L.

However, when considering general deductions of the form $\sigma_1, \ldots, \sigma_n \vdash \sigma$ (where the $\sigma_i$ may also be partial correctness specifications), it is easy to come up with semantically valid deductions which cannot be derived in H. Two examples are

1. $p \mid if \ r \ then \ \alpha \ else \ \beta \ fi \ q \ \vdash \ p \mid if \ \neg \ r \ then \ \beta \ else \ \alpha \ fi \ q$

2. $p \mid \{a\}q, r \{a\}q \vdash pvr\{a\}q$

(a rule which, while being underivable in H, can be shown to be superfluous for any concrete proof of partial correctness, Igarashi et al[12]).

These examples illustrate the absence (in H) of mechanisms for (1) extracting information from a specification $p\{a\}q$ about parts of $\alpha$ (where $\alpha$ is a complex program segment), and (2) combining the information given in different specifications about the same program segment. H can be seen to be complete only for "simple" deductions, in which the antecedents $\sigma_i$ include for each given $\alpha$, at most one specification of the form $p\{a\}q$, and all such $\alpha$'s are simple specifications consisting of a single assignment or call statement, or a single program segment variable (PSV), which is a symbol standing for an arbitrary program segment.

In Section II we present our system $D$ which is an extension of Hoare's system, and in Section III show that $D$ is sound and complete for any deduction $(\sigma_1, \ldots, \sigma_n) \vdash D \sigma$ iff $\sigma_1, \ldots, \sigma_n \vdash \sigma$, that is, $\sigma$ can be proved in $D$ from assumptions $\sigma_1, \ldots, \sigma_n$, iff $\sigma$ is true in every model satisfying $\sigma_1, \ldots, \sigma_n$. Here the $\sigma_i$ can themselves be any partial correctness specifications.

The completeness result is shown by proving a series of more
restricted theorems, holding for successively richer subsystems of $D$
and thus clarifying the whole process and also achieving a side effect of
indicating the precise role in $D$ played by its important components.

A variety of properties of programs can be proved using $D$,
and the completeness result ensures us that when $L$ is expressive (e.g. in
arithmetic), a proof exists for each valid such property. The following
possibilities are described in Section IV:

(i) proving the partial correctness of a given
    program,

(ii) proving the total correctness of a given
     program,

(iii) proving the (strong) equivalence of programs,

(iv) establishing derived rules,

(v) carrying out modular proofs of program
    correctness given properties of segments of
    the program,

(vi) simplifying complex program segments and
    establishing valid program transformations.

Schematically speaking, $D$ will consist of a suitable version
of $H$ for composing the conclusion of the deduction, four rules
($\land, \lor, \exists, \forall$) for collecting information about unspecified program segments,
and a "mirror image" of $H$ containing inverse rules for decomposing
complex program segments appearing among the premises. $D$, having the
flavour of a natural deduction system, has all valid formulae of $L$ as
axioms.

II. The System.

Syntax

The alphabet $\Sigma$ contains symbols for individual constants and
variables, functions and predicates, connectives and operators.
$L=L(\Sigma)$ is a logical language with equality over $\Sigma$ (having at
least the power of the first order language over $\Sigma$). A
well-formed formula of $L$ will be called a logical wff ($L$-wff). $P=P(\Sigma)$
is a programming language over $\Sigma$, with the following syntax:
<statement>::=<elementary action> | <procedure call> |
<statement>;<statement> |
if<bool> then<statement> else<statement> fi |
while<bool> do<statement> od

declaration>::=<procedure name>({<name parameter list>},
<value parameter list>) proc<statement> end.

An elementary action is a non deterministic assignment of the form
\( x \leftarrow x' A(x, x', y) \) reading: "assign to \( x \) some \( x' \) such that \( A \)
holds". This will usually be abbreviated as \( A(x, y) \), where \( x \) is the vector
of variables which can be modified by \( A \), and \( y \) is the vector of additional
variables upon which the assignment might depend. When \( A \) is of the
form \( x' = t(x, y) \), \( A \) is the conventional assignment statement.

A procedure call is a statement of the form call \( P(x, t) \), where \( P \)
is a procedure name, \( x \) is a vector of actual name parameters (variables), and
\( t \) is a vector of actual value parameters (terms). The \( x' \)'s are assumed to be
distinct and the \( t \)'s to be independent of the \( x' \)'s.

A boolean is a quantifier-free L-uff.

A P-segment will simply be a statement in \( P \). We extend \( \Sigma \) to
\( \Sigma' \) by adding a set of new symbols \( (R_1, R_2, \ldots) \) which
stand for arbitrary P-segments, and are therefore called program-segment
variables (PSV's). The programming language \( P' \) is an extension of \( P \)
obtained by allowing statements of the form \( R_i(x, y) \), where \( x \) and \( y \)
have a meaning similar to that given in the elementary actions. Note that
the difference between a PSV and an elementary action is that for the
latter we are given a formula defining its effect. Similarly, the
difference between a PSV and a procedure call is that the latter may have
an explicit declaration. We will use \( \alpha(x, y) \) to denote an arbitrary
P'-segment such that \( x \) is the vector of all modifiable variables of \( \alpha \), and
\( y \) consists of all other variables appearing in \( \alpha \).

A specification is a construct \( \sigma \) of the form
\( \sigma : p(x, y, v) \mid (x, y) \mid q(x, y, v) \), where \( p \) and \( q \) are L-uffs and \( \alpha \) is a P'-segment.
Here the elements of \( \nu \) are said to be the free variables of the
specification \( \sigma \). Where no confusion can arise we will occasionally
omit the \( x' \)'s and regard the \( y \) as consisting of all the variables appearing
in the specification not assigned to in \( \alpha \). A specification \( p \mid (x, y) \mid q \) is simple
if \( \alpha \) is a PSV, an elementary action or a call statement
(simple statements).

The formulae of our language \( \mathcal{W} \) (called \( \mathcal{W} \)-uffs) are

(1) L-uffs,
(2) specifications,
(3) declarations.
(Note that $W$-wffs cannot be combined by logical connectives.)

**Semantics**

An interpretation of a set $\Gamma$ of $W$-wffs is a tuple $I = \langle D, \Sigma, R_1, \ldots, R_k, P_1, \ldots, P_m, \rangle$,

where $D$ is a nonempty domain, $\Sigma$ is an interpretation of all individuals (including constants and free variables), function and predicate symbols of $L$, each $R_i(x, x', y)$ is a relation for a PSV $R_i(x, y)$ appearing in $\Gamma$, and each $P_i(x, x', y)$ is a relation for a procedure $P_i$ that appears in $\Gamma$, but does not have a corresponding declaration in $\Gamma$.

$R_i$ and $P_i$ describe the effect of the $P'$-segments $R_i$ and call$P_i$, respectively, under $I$.

We now show how an interpretation $I$ assigns truth values to $W$-wffs. An $L$-wff is assigned a truth value by $I$ in the standard way. A program segment $\alpha(x, y)$ all of whose procedure calls are interpreted (see below), is interpreted under $I$ as a relation $\rho_{\alpha}$ in the following way (relational notation from e.g. deBakker and Meertens[1]):

For an elementary action $A$ $\rho_A = A$ (the interpretation $\Sigma$ gives to $A$),

For a PSV $R_i$ $\rho_{R_i} = R_i$

For a procedure $P_i$ $\rho_{\text{call}P_i} = P_i$

$\rho_{\alpha; \beta} = \rho_{\alpha} \rho_{\beta}$

$\rho_{\text{if } r \text{ then } \alpha \text{ else } \beta} = \rho_{\alpha} \cup \rho_{\beta}$

$\rho_{\text{while } r \text{ do } \alpha} = (\rho_{\alpha})^* r'$.

Using this definition, we are now able to assign relations to the procedure calls which have corresponding body declarations in $\Gamma$. The relations assigned to these procedures are the least fixpoint relations solving the system of mutually recursive procedure declarations in $\Gamma$ (here too we will refer to this interpretation of such $P$ as $P$). We now have an interpretation under $I$ for each $P'$-segment in $\Gamma$.

A specification $p(x, y, z) \alpha(x, y) \land q(x, y, z) \beta(x, y)$ is true under $I$ if $\forall x, y (p(x, y, z) \land \rho_{\alpha}(x, y) > q(x, y, z))$ is true (note that the free variables $y$ have been assigned by $I$).

A set $\Gamma$ of $W$-wffs is defined to be true under an interpretation $I$ of $\Gamma$, if all non-declaration formulae of $\Gamma$ are
true in \( I \). \( I \) is called a **model** of \( \Gamma \).

A tuple \( S = (\sigma_1, \ldots, \sigma_n, \sigma) \) where \( \sigma \) is not a declaration, is called a **valid deduction** (written \( \sigma_1, \ldots, \sigma_n \vdash \sigma \)), if \( \sigma \) is true in any interpretation \( I \) of \( S \) which is a model of \( \{\sigma_1, \ldots, \sigma_n\} \).

We denote a \( P' \)-segment \( \alpha \) containing the statements \( \mathsf{callP}_1(x_1, t_1), \ldots, \mathsf{callP}_n(x_n, t_n) \) by \( \alpha(\text{"callP}_1", \ldots, \text{"callP}_n") \), and the elementary action \( \mathsf{a} \mid \mathsf{b} \cdot (\phi(x, t) \circ \psi(x', t')) \) by \( [\phi, \psi](x, t) \).

We now present our system \( D \). The basic statements to be proved in \( D \) are deductions of the form \( \Gamma \vdash \sigma \) where \( \Gamma \) is a set of \( W \)-wffs, and \( \sigma \) a non-declaration \( W \)-wff. Our inference rules are rule-schemata in which \( \alpha, \beta, \ldots \) stand for arbitrary \( P' \)-segments and \( p, q, \ldots \) for arbitrary \( L \)-wffs.

**AXIOMS**

\( A_1 \) \( \vdash p \)

where \( p \) is any logically valid \( L \)-wff.

\( A_2 \) \( \Gamma, \sigma \vdash \sigma \)

where \( \sigma \) is not a declaration.

\( A_3 \) Frame axiom

\( \vdash p(y) \{\alpha(x,y)\} p(y) \)

where \( y \) and \( x \) are disjoint.

**RULES OF INFERENCE**

\( L_1 \) Introduction

\[
\Gamma \vdash \sigma_1 \\
\hline
\Gamma, \sigma_2 \vdash \sigma_1
\]

\( L_2 \) Modus Ponens

\[
\Gamma \vdash \sigma_1, \Gamma \vdash \sigma_2 \\
\hline
\Gamma \vdash \sigma_1 \rightarrow \sigma_2
\]

\( L_3 \) Simplification

\[
\Gamma \vdash \sigma_1, \Gamma \vdash \sigma_2 \\
\hline
\Gamma \vdash \sigma_1 \land \sigma_2
\]

\( L_4 \) Conjunction

\[
\Gamma \vdash \sigma_1 \\
\Gamma \vdash \sigma_2 \\
\hline
\Gamma \vdash \sigma_1 \land \sigma_2
\]

\( L_5 \) Implication

\[
\Gamma \vdash \sigma_1, \Gamma \vdash \sigma_2 \\
\hline
\Gamma \vdash \sigma_1 \rightarrow \sigma_2
\]

\( L_6 \) Disjunction

\[
\Gamma \vdash \sigma_1 \\
\Gamma \vdash \sigma_2 \\
\hline
\Gamma \vdash \sigma_1 \lor \sigma_2
\]

\( L_7 \) Negation

\[
\Gamma \vdash \sigma_1 \\
\hline
\Gamma \vdash \neg \sigma_1
\]

\( L_8 \) Universal Quantification

\[
\Gamma \vdash \sigma_1 \quad \forall x \phi(x) \\
\hline
\Gamma \vdash \phi(x)
\]

\( L_9 \) Existential Quantification

\[
\Gamma \vdash \phi(x) \\
\hline
\Gamma \vdash \exists x \phi(x)
\]

\( L_{10} \) Equality

\[
\Gamma \vdash \phi(x) \\
\Gamma \vdash \phi(y) \\
\hline
\Gamma \vdash \phi(x) \leftrightarrow \phi(y)
\]

\( L_{11} \) Substitution

\[
\Gamma \vdash \phi(x) \\
\hline
\Gamma \vdash \phi(y)
\]

\( L_{12} \) Normal Form

\[
\Gamma \vdash \phi(x) \\
\hline
\Gamma \vdash \phi(x')
\]

\( L_{13} \) Skolemization

\[
\Gamma \vdash \phi(x) \\
\hline
\Gamma \vdash \phi(y)
\]

\( L_{14} \) Case Analysis

\[
\Gamma \vdash \phi(x) \\
\Gamma \vdash \phi(y) \\
\hline
\Gamma \vdash \phi(x) \lor \phi(y)
\]

\( L_{15} \) Generalization

\[
\Gamma \vdash \phi(x) \\
\hline
\Gamma \vdash \phi(y)
\]

\( L_{16} \) Specialization

\[
\Gamma \vdash \phi(x) \\
\hline
\Gamma \vdash \phi(y)
\]

\( L_{17} \) Resolution

\[
\Gamma \vdash \phi(x) \\
\Gamma \vdash \phi(y) \\
\hline
\Gamma \vdash \phi(x) \land \phi(y)
\]

\( L_{18} \) Unification

\[
\Gamma \vdash \phi(x) \\
\hline
\Gamma \vdash \phi(y)
\]

\( L_{19} \) Abstraction

\[
\Gamma \vdash \phi(x) \\
\hline
\Gamma \vdash \lambda x \phi(x)
\]

\( L_{20} \) Application

\[
\Gamma \vdash \phi(x) \\
\Gamma \vdash \psi(y) \\
\hline
\Gamma \vdash \phi(x) \psi(y)
\]

\( L_{21} \) Function Composition

\[
\Gamma \vdash \phi(x) \\
\Gamma \vdash \psi(y) \\
\hline
\Gamma \vdash \phi(x) \circ \psi(y)
\]

\( L_{22} \) Function Evaluation

\[
\Gamma \vdash \phi(x) \\
\Gamma \vdash \psi(y) \\
\hline
\Gamma \vdash \phi(x, y)
\]

\( L_{23} \) Function Composition

\[
\Gamma \vdash \phi(x) \\
\Gamma \vdash \psi(y) \\
\hline
\Gamma \vdash \phi(x) \circ \psi(y)
\]

\( L_{24} \) Function Evaluation

\[
\Gamma \vdash \phi(x) \\
\Gamma \vdash \psi(y) \\
\hline
\Gamma \vdash \phi(x, y)
\]
L2 Modus Ponens
\[ \Gamma, \sigma_2 \vdash \sigma_1, \Gamma \vdash \sigma_2 \]
\[ \Gamma \vdash \sigma_1 \]

L3 Deduction
\[ \Gamma, p \vdash q \quad \text{and} \quad \Gamma \vdash p, \text{wcq} \]
\[ \Gamma \vdash p, \text{wcq} \quad \Gamma, p \vdash q \]
where p and q are L-wwfs.

D1 Elementary Action
\[ \Gamma \vdash p(x, u, y) \land \hat{A}(x, x', u) \supset q(x', u, y) \]
\[ \Gamma \vdash p(x, u, y) \land (\hat{A}(x, u) \land q(x, u, y)) \]

D2 Consequence
\[ \Gamma \vdash p(s \cdot r), \Gamma \vdash s \cdot r, \Gamma \vdash r, \text{wcqp} \]
\[ \Gamma \vdash p \cdot (a \cdot q) \]

D3 Composition
\[ \Gamma \vdash p(\alpha) \cdot s, \Gamma \vdash \cdot s(\beta \cdot q) \]
\[ \Gamma \vdash p(\alpha; \beta) \cdot q \]

D4 Conditional
\[ \Gamma \vdash p, \text{wc}(\alpha) \cdot q, \Gamma \vdash p, ^\land \text{wc}(\beta) \cdot q \]
\[ \Gamma \vdash p, \text{if r then } \alpha \text{ else } \beta \text{ fi} \cdot q \]

D5 Iteration
\[ \Gamma \vdash p, \text{while } r \text{ do } \alpha \text{ od} \cdot p \land ^\land \cdot r \]
D6 Substitution

\[ \Gamma \vdash p(x, y) \{ \alpha(x, u) \} q(x, y) \]
(a)
\[ \Gamma \vdash p(z, u) \{ \alpha(z, u) \} q(z, u) \]

where \( z \) is disjoint from \( u \) and the \( z \)'s are distinct and do not appear in \( p \) and \( q \).

\[ \Gamma \vdash p(x, y) \{ \alpha(x, y) \} q(x, y) \]
(b)
\[ \Gamma \vdash p(x, t) \{ \alpha(x, t) \} q(x, t) \]

where \( t \) is a vector of terms which is free for \( u \) in \( p \) and \( q \), and does not depend on \( x \).

D7 Recursion

\[ \Gamma([\phi, \psi]) \vdash \phi(\{\alpha([\phi, \psi])\}) \psi \]

(Here \( \Gamma([\phi, \psi]) \) is \( \Gamma \) with the elementary action \( [\phi, \psi](z, t) \) substituted for occurrences of \( \text{callP}(z, t) \). A clarification of rule D7 appears at the end of the Section.)

D8 \&-rule

\[ \Gamma \vdash p(\alpha) q_1, \ldots, \Gamma \vdash p(\alpha) q_n \]

\[ \Gamma \vdash p(\alpha) \wedge q_i \]

\[ \Gamma \vdash p(\alpha) \quad n \geq 0 \]

\[ i=1 \]

D9 \lor-rule

\[ \Gamma \vdash p_1(\alpha) q_1, \ldots, \Gamma \vdash p_n(\alpha) q_n \]

\[ \Gamma \vdash \lor p_i(\alpha) q_i \quad n \geq 0 \]

\[ i=1 \]

(D8 and D9 reduce to \( \Gamma \vdash p(\alpha) \text{true} \) and \( \Gamma \vdash p(\alpha) \text{false} \) respectively when \( n=0 \).

D10 \forall-rule

\[ \Gamma \vdash p(\alpha) q \]

\[ \Gamma \vdash p(\alpha)(\forall y) q \]
y not free in p or Γ, and does not appear in α.

D11  ξ-rule

\[ Γ \vdash p(\alpha)q \]

\[ Γ \vdash (\exists y)p(\alpha)q \]

y not free in q, and does not appear in α.

D12  Inverse Elementary Action

\[ Γ, p(x, u, y) \land A(x, x', y) \supset q(x', u, y) \vdash σ \]

\[ Γ, p(x, u, y) \land A(x, y)q(x, u, y) \vdash σ \]

D13  Inverse Composition

\[ Γ, p(\alpha; λ, λ(\beta)q \vdash σ \]

\[ Γ, p(\alpha; β)q \vdash σ \]

where λ does not appear in any other component of the rule.

Note that D13 (and similarly for the other inverse rules) is an indirect way of expressing the more natural

\[ Γ \vdash p(\alpha; β)q \]

\[ Γ \vdash ξ p(\alpha) λ, λ(\beta)q \]

the conclusion of which is unfortunately not well formed in \( W \).

D14  Inverse Conditional

\[ Γ, p \land (\alpha)q, p \land (β)q \vdash σ \]

\[ Γ, p(\text{if } r \text{ then } α \text{ else } β \text{ if})q \vdash σ \]

D15  Inverse Iteration

\[ Γ, p \alpha, λ(\alpha)λ, λ \rightarrow cp \vdash σ \]

\[ Γ, p(\text{while } r \text{ do } α \text{ end})q \vdash σ \]
where $\lambda$ does not appear in any other component of the rule.

D16 Inverse Recursion

$$\Gamma([\delta, \lambda]), \delta \alpha([\delta, \lambda]) \lambda \vdash \sigma$$

$$\Gamma("callP"), \text{P proc} \alpha("callP")\text{end} \vdash \sigma$$

where $\delta$ and $\lambda$ do not appear in any other component of the rule.

A proof in $D$ is a sequence of deductions $\Gamma_1 \vdash \sigma_1, \ldots, \Gamma_2 \vdash \sigma_2$, where any line (i.e., deduction) is an axiom or is derived from previous lines by one of the inference rules. A deduction $\Gamma \vdash \sigma$ is said to be derivable in $D$ (written $\Gamma \vdash_D \sigma$) if it is a line of a proof in $D$.

Our formulation of D7 employs the substitution of $[\phi, \psi]$ for "callP" in the proof of the body $\alpha$. This corresponds to the familiar notion of assuming $\phi\{\text{callP}\}\psi$ when proving $\alpha$. Employing the same substitution for the premises used in proving $\alpha$, provides us with a concise way of constructing a recursion rule for mutually recursive procedures which avoids referring to all $n$ procedures (as is done in [7] and [3]). In order to illustrate the way in which D7 (and similarly D16) is used, consider two procedures $P_1$ and $P_2$, with declarations $P_i \text{proc} \alpha_i("callP_1", "callP_2")\text{end} 1 \leq i \leq 2$. A framework for a proof of a $\text{callP}_1$-specification is:

1. $\vdash \phi_2 \alpha_2([\phi_1, \psi_1], [\phi_2, \psi_2]) \psi_2$.
2. $P_2 \text{proc} \alpha_2([\phi_1, \psi_1], "\text{callP}_2")\text{end} \vdash \phi_2 \{\text{callP}_2\} \psi_2$.
3. $P_2 \text{proc} \alpha_2([\phi_1, \psi_1], "\text{callP}_2")\text{end} \vdash \phi_1 \alpha_1([\phi_1, \psi_1], \text{callP}_2) \psi_1$.
4. $P_1 \text{proc} \alpha_1 \text{end}, P_2 \text{proc} \alpha_2 \text{end} \vdash \phi_1 \{\text{callP}_1\} \psi_1$.

Lines (2) and (4) are proved using D7 with an empty $\Gamma$, and $\Gamma$ consisting of the $P_2$-declaration respectively.

The following (standardly verified) fact is very useful in proving deductions involving unspecified program segments:

**Substitution Theorem** - If $\Gamma \vdash_D \sigma$, and $\Gamma'$ and $\sigma'$ are obtained by replacing all occurrences of a PSV $R(x, y)$ by...
an arbitrary $P'$-segment $\alpha(x,y)$ in $\Gamma$ and $\sigma$, then $\Gamma' \vdash_D \sigma'$.

III. Results.

One of our basic assumptions throughout, is that the language $L$ is expressive (Cook [3]). This means that for each $P$-segment $\alpha$ in the context of a given set of declarations, it is possible to express as an $L$-wff the relation $\rho_\alpha$ computed by $\alpha$, i.e., $L$ has constructs powerful enough to express the $\ast$, $U$, composition and fixpoint operators. A special important case of an expressive language is (as pointed out by Cook) first order arithmetic.

All subsystems considered in this section have A1-A3 as axioms, L1-L3 as logical rules and differ only in their D-rules.

Consider the system $D1$ which consists of rules D1-D5. $D1$ is a version of the usual Hoare system for proving partial correctness of programs with regular control structure, and for it we have the following result (proved e.g. in [1,3,9,15]):

Theorem 1 - If $\sigma_1,\ldots,\sigma_n$ are $L$-wffs, then

$\sigma_1,\ldots,\sigma_n \vdash \sigma$ iff $\sigma_1,\ldots,\sigma_n \vdash_{D1} \sigma$.

Consider $D2$ consisting of D1-D7. This is an extension of Hoare's method to deal with mutually recursive procedures. A proof of Theorem 2 can be found for similar versions in Harel et al [3] or Gurevich [7].

Theorem 2 - If $\sigma_1,\ldots,\sigma_n$ are $L$-wffs or procedure declarations, then

$\sigma_1,\ldots,\sigma_n \vdash \sigma$ iff $\sigma_1,\ldots,\sigma_n \vdash_{D2} \sigma$.

We now consider $D3$ which consists of rules D1-D6 and D8-D12.

Theorem 3 - If $\sigma_1,\ldots,\sigma_n$ are $L$-wffs or simple specifications, then

$\sigma_1,\ldots,\sigma_n \vdash \sigma$ iff $\sigma_1,\ldots,\sigma_n \vdash_{D3} \sigma$.

(Note - in this and the following theorems we omit the proofs of the soundness direction. The reader is urged to convince himself that the
rules are indeed sound, a rigorous proof of this would be based on Scott's induction principle in a standard way. Rather, the proofs presented are designed to demonstrate the completeness direction in a constructive manner.

Proof - Given a valid deduction \( \sigma \vdash \sigma \)

we reduce the problem as follows:

1. The absence of procedure declarations among the premises means that each call statement can be regarded as a new PSV. This follows from the arbitrary interpretations both PSV's and call's can take on, in a model of \( \sigma \).

2. Use rule D12 to replace every elementary-statement specification \( p | A | q \) by an L-wff. (Here, as well as at other points in the paper, we describe the natural order of the derivation. Formally, this application of rule D12, for example, appears at the end of the proof in \( D \). Nevertheless, we may think of this stage as being first in the derivation process.) We are left with premises consisting of PSV-specifications of the form \( p | R | q \) and L-wffs. Denote by \( \tau \) the conjunction of the latter. Formally, \( \tau \) can be derived by using D8 with the identity program and \( p \text{true} \).

3. If \( \sigma \) is an L-wff then the validity of the deduction \( \omega \) is equivalent to the validity of some L-wff. This can be seen by considering an interpretation in which each PSV is assigned the empty relation. In this case all specification premises hold and therefore we must have \( \tau \vdash \sigma \), which is equivalent to the validity of \( \tau \vdash \sigma \), which in turn is an axiom in \( A1 \). Using L2, \( \sigma \) is obtained.

4. Employing a similar argument with an interpretation assigning the empty relation to all PSV's not appearing in \( \sigma \), we can omit any PSV-specification for a PSV not appearing in \( \sigma \). We are now left with a situation of the form

\[ \tau, R_1\text{-specifications}, \ldots, R_k\text{-specifications} \vdash p(\alpha(R_1, \ldots, R_k))q \]

where \( \alpha \) is a \( \sigma \)-segment involving PSV's \( R_1, \ldots, R_k \).

Denote the specification premises by \( \Gamma \). These premises contain all available information about \( R_1, \ldots, R_k \). We therefore construct for each \( \Gamma \) an "approximation from above" \( \mu_{R_i} \) to the relation computed by \( R_i \).

\( \mu_{R_i} \) will be an L-wff which can easily be seen to be true in any model of \( \Gamma \), and hence in any model of \( \{ \tau, \Gamma \} \). This is the sense in which it is an approximation. We will simplify notation by referring to the case where \( k = 1 \) and to \( R_1 \) as \( R \), with the understanding that the following can be done for all \( k \), PSV's for any \( k \).

Assume that \( \Gamma \) is the set \( p_j(x, y) (R(x, y))q_j(x, y) \) \( 1 \leq j \leq m \).

This can be brought about by using D6 and collecting free variables in \( y \).

Define
$$\mu_R(x, x', y) = \forall y \land (p_j(x, y', y) \supset q_j(x', y', y))$$

Clearly $\mu_R$ serves to "collect information" about the PSV $R$.

Define $A_R$ as the elementary action $x \Leftarrow x' \mu_R(x, x', y)$. Obviously $\mu_{A_R} = \mu_R$. From the way $A_R$ was defined, it is clear that for every $j$ we have $\models p_j(A_R)q_j$. Thus under the substitution that replaces the PSV $R$ by the $P$-segment $A_R$, every interpretation satisfying $\tau$ also satisfies $\Pi$, and therefore also satisfies $p(\alpha(A_R))q$. Hence $\tau \models p(\alpha(A_R))q$, and by Theorem 1 there exists a proof

($)\quad \tau \vdash_{D1} p(\alpha(A_R))q.$

Without loss of generality (having in mind the standard techniques used in proving Theorem 1, in e.g. [1, 3, 9, 15]), we may assume that in the process of proving the deduction ($\ast$) in $D1$, the strongest consequent approach was adopted, in which every subderivation of a simple $A_R$-specification is preceded by a derivation of a specification of the form $s(A_R)s \cdot \mu_R$ for some $s$, where for $s(x, y)$ and $\mu_R(x, x', y)$ we define $s \cdot \mu_R(x, y) = \exists x'(s(x', y) \land \mu_R(x', x, y))$. (See e.g. [1]).

If we now manage to replace every such subproof by a proof in $D3$ of $s(\alpha(R)s \cdot \mu_R$ from assumptions $\Pi$ and substitute $R$ for $A_R$ elsewhere, then this modified proof of ($\ast$) serves as our proof of $\Pi, \tau \vdash_{D3} p(\alpha(R))q$. Indeed this can be done using the following four derived rules of $D3$:

**D8'** $\land$-rule

$$\Gamma \vdash p_1\{\alpha\}q_1, \ldots, \Gamma \vdash p_n\{\alpha\}q_n$$

$$\begin{array}{c}
\Gamma \vdash \bigwedge_{i=1}^{n} p_i\{\alpha\} \land q_i \\
\Gamma \vdash \bigwedge_{i=1}^{n} p_i\{\alpha\} \land q_i
\end{array}$$

**D9'** $\lor$-rule

$$\Gamma \vdash p_1\{\alpha\}q_1, \ldots, \Gamma \vdash p_n\{\alpha\}q_n$$

$$\begin{array}{c}
\Gamma \vdash \bigvee_{i=1}^{n} p_i\{\alpha\} \lor q_i \\
\Gamma \vdash \bigvee_{i=1}^{n} p_i\{\alpha\} \lor q_i
\end{array}$$
010. ∀y-rule

\[ \Gamma \vdash p \alpha \lambda q \]
\[ \Gamma \vdash (\forall y) p \alpha \lambda (\forall y) q \]

\( y \) not free in \( \Gamma \), and does not appear in \( \alpha \).

011. ∃-rule

\[ \Gamma \vdash p \alpha \lambda q \]
\[ \Gamma \vdash (\exists y) p \alpha \lambda (\exists y) q \]

\( y \) does not appear in \( \alpha \).

Now (for any \( s \)) replace every subproof of
\[ s \vdash p \alpha \lambda q \]

in the proof of \( (x) \) by:

\[ \Gamma \vdash p_j (x', y, y) \circ p_j (x, y, y) \circ \{ R(x, y) \} p_j (x', y, y) \circ p_j (x, y, y) \]

for every \( 1 \leq j \leq m \). (Use A3 with \( \neg p_j (x', y, y) \), and D9')

\[ \Gamma \vdash \forall x (p_j (x', y, y) \circ p_j (x, y, y) \circ \{ R(x, y) \} p_j (x', y, y) \circ p_j (x, y, y)) \]

\[ \forall x (p_j (x', y, y) \circ p_j (x, y, y)) \] (D10')

\[ \Gamma \vdash s(x', y) \wedge \forall x (p_j (x', y, y) \circ p_j (x, y, y) \circ \{ R(x, y) \} p_j (x', y, y) \circ p_j (x, y, y)) \]

\[ j = 1 \]

\[ s(x', y) \wedge \forall x (p_j (x', y, y) \circ p_j (x, y, y)) \]

\[ j = 1 \]

(Use A3 with \( s(x', y) \), and D9')

\[ \Gamma \vdash \exists x' (s(x', y) \wedge \forall x (p_j (x', y, y) \circ p_j (x, y, y) \circ \{ R(x, y) \} p_j (x', y, y) \circ p_j (x, y, y))) \]

\[ j = 1 \]

\[ \exists x' (s(x', y) \wedge \forall x (p_j (x', y, y) \circ p_j (x, y, y))) \] (D11')

\[ \exists x' (s(x', y) \wedge \forall x (p_j (x', y, y) \circ p_j (x, y, y))) \]

\[ j = 1 \]

(Al and L1)

\[ \Gamma \vdash s(x, y) \circ \exists x' (s(x', y) \wedge \forall x (p_j (x', y, y) \circ p_j (x, y, y))) \]

\[ j = 1 \]

(A1 and L1)

\[ \Gamma \vdash s(x, y) \circ R(x, y) \circ s \circ R(x, y) \] (D2).
derived in $D4$ from $\tau$ and $\Pi$, where $x_0$ is a vector of new symbols. Easy applications of $A3, D8'$ and $D11'$ will give $s|\text{call}\,P_i|s - \mu_{P_i}$.

We prove that $\Pi, \tau \vdash_{D4} x = x_0 \{\text{call}\,P_i\} \mu_{P_i}(x_0, x, y)$ by induction on $m$. For $m = 1$ assume that if $\Pi$ contains $m-1$ declarations $P_1, \ldots, P_{m-1}$ (denoted $\Pi^{(m-1)}$), then for every $1 \leq i \leq m-1$

$\Pi^{(m-1)}, \tau \vdash_{D4} x = x_0 \{\text{call}\,P_i\} \mu_{P_i}(x_0, x, y)$.

Given $\Pi^{(m)}$, consider the first $m-1$ declarations with $A_{P_m}$ substituted for "call\,P_m" (denote this by $\Pi^{(m-1)}(A_{P_m})$). It is not difficult to see that

$\tau, \{x = x_0 \{\text{call}\,P_i\} \mu_{P_i}(x_0, x, y) \mid 1 \leq i \leq m-1\} \vdash_{D3} x = x_0 \{\alpha_{P_m}(A_{P_m})\} \mu_{P_m}(x_0, x, y)$, hence by Theorem 3

$\tau, \{x = x_0 \{\text{call}\,P_i\} \mu_{P_i}(x_0, x, y) \mid 1 \leq i \leq m-1\} \vdash_{D3} x = x_0 \{\alpha_{P_m}(A_{P_m})\} \mu_{P_m}(x_0, x, y)$.

However by the inductive hypothesis, for every $1 \leq i \leq m-1$

$\tau, \Pi^{(m-1)}(A_{P_m}) \vdash_{D4} x = x_0 \{\text{call}\,P_i\} \mu_{P_i}(x_0, x, y)$.

We therefore have

$\tau, \Pi^{(m-1)}(A_{P_m}) \vdash_{D4} x = x_0 \{\text{call}\,P_i\} \mu_{P_i}(x_0, x, y)$,

and applying rule $D7$ we obtain

$\tau, \Pi^{(m)} \vdash_{D4} x = x_0 \{\text{call}\,P_m\} \mu_{P_m}(x_0, x, y)$. 

The process described in the last two theorems can be summarized as a process for "composing" a complex conclusion from simple premises. We now begin the process of "decomposing" complex premises.

Consider $D5$, consisting of rules $D1-D15$.

Theorem 5 - If $\sigma_1, \ldots, \sigma_n$ are as in Theorem 4 without the requirement that specifications be simple, then

$\sigma_1, \ldots, \sigma_n \vdash \sigma$ iff $\sigma_1, \ldots, \sigma_n \vdash_{D5} \sigma$.

Proof - All non-simple specifications among the premises are decomposed
We remark here that restricting the premises to have no free variables not appearing in $\alpha$ (i.e. no $y$), makes possible a different proof of Theorem 3 which does not use rules D10-D11.

We now consider $D4$ consisting of D1-D11.

**Theorem 4** - If $\sigma_1, \ldots, \sigma_n$ are L-wffs, simple specifications or declarations for procedure names not appearing in these simple specifications (but possibly in $\sigma$, and in other declarations), then

$$\sigma_1, \ldots, \sigma_n \vdash \sigma \iff \sigma_1, \ldots, \sigma_n \vdash D4 \sigma.$$  

**Proof** - Assume given $w: \sigma_1, \ldots, \sigma_n \vdash \sigma$, with procedure declarations $P_i \proc \alpha_i \end 1i \leq m$, among the premises. $D4$ illustrates the extra feature of call's (in $\sigma$) to procedures with given bodies, thus forcing the use of D7. We will find a similar approximation $\mu_{P_i}$ for each such procedure. As before regard each call to a procedure other than the $P_i$'s as a new PSV. We now construct $\mu_R$ for every PSV $R$, and as above, substitute $A_R$ for each appearance of $R$ in $w$. Denote the resulting modified body of $P_i$ by $\alpha'_i$, and modified $\sigma$ by $\sigma'$.

This system of $m$ PSV-free declarations now gives rise to a least-fixpoint solution, in the form of $m$ relations. Denote the L-wff equivalents to these relations by $\mu_{P_i} 1i \leq m$. Define $A_{P_i}$ to be the elementary action

$$A_{P_i} \equiv (x, x', y) \cdot (\mu_{P_i} (x, x', y)).$$

(For clarity throughout this proof we omit indices of $x, x'$ and $y$.) Denoting as before by $\tau$ and $I$ the L-wff and specification premises respectively, we now observe that any interpretation $I$ satisfying $\tau$, satisfies (substituted) $I$.

Recalling the definition of the relation that $I$ assigns to each $P_i$ we have $\tau \vdash \sigma''$, where $\sigma''$ is $\sigma'$ further modified by substituting $A_{P_i}$ for $\text{call} P_i \text{, } 1i \leq m$. Therefore there exists a proof

$$\tau \vdash D1 \sigma''.$$  

Denoting the declaration premises of $w$ by $\Pi$, we will obtain a proof of $w$ in $D4$ by first replacing (in the proof of (**)) subproofs of

$$\tau \vdash D1 \quad s \{ A_{P_i} \} \text{ by}$$

proofs of

$$s \{ \text{call} P_i \} \text{ by}$$

and then dealing with PSV's as in

Theorem 3. We will really show how $x = x_0 \{ \text{call} P_i \} \mu_{P_i} (x_0, x, y)$ can be
IV. The Power of \( \mathcal{D} \).

We will try to be slightly more specific about our claims as to what can be done in \( \mathcal{D} \).

(i) (partial correctness) Given a program \((P_1, \ldots, P_n, \alpha)\) consisting of \(n\) declarations and a statement \(\alpha\), and some \(L\)-wffs \(\tau_1, \ldots, \tau_m\), a proof that the program is partially correct with respect to \(p\) and \(q\), assuming the \(\tau_i\) are true, is carried out simply by proving in \(\mathcal{D}\)

\[ P_1, \ldots, P_n, \alpha, \tau_1, \ldots, \tau_m \vdash p(\alpha) \wedge q. \]

(ii) (total correctness), given a program and \(L\)-wffs as in (i), a proof that \(\alpha\) is totally correct assuming the \(L\)-wffs true, can be carried out by proving in \(\mathcal{D}\)

\[ P_1, \ldots, P_n, \alpha, \tau_1, \ldots, \tau_m, \lambda(x, y) (\alpha(x, y)) \neg q(x, y, y) \vdash \forall x, y (p(x, y, y) \supset \neg \lambda(x, y)). \]

where \(\lambda\) is a new predicate symbol.

Another way is by using constant symbols \((a, b)\) and proving in \(\mathcal{D}\)

\[ P_1, \ldots, P_n, \alpha, \tau_1, \ldots, \tau_m, p(a, b, x), (a, b) = (x, y) (\alpha(x, y)) \neg q(x, y, y) \vdash false. \]

We wish to clarify this somewhat surprising result as related to the commonly accepted view that termination of programs with loops or recursion must employ some form of induction on a well-founded set. The fact is that the induction has been buried deep in \(L\), and its utilization is no longer the concern of the user of \(\mathcal{D}\). Rather, an inductive argument might be handy when the valid formulae (taken by us as axioms \(A_1\)) are to be proved in \(L\). We illustrate this point. Take \(L\) to be the language of arithmetic, and prove that \(\alpha\): while \(x > 0\) do \(x \leftarrow x - 1\) od is totally correct with respect to \(p(x) \equiv x > 0\) and \(q(x) \equiv x = 0\). Subgoal (using the second formulation above), we obtain the deduction \(a \geq 0, x = a d \leftarrow x \leftarrow 0 \vdash false\). Applications of D15 and D12 yield \(a \geq 0, \forall x (x = a \circ \lambda(x)), \forall x (\lambda(x) \wedge x \leq 0 \supset x = 0)\), \(\forall x (\lambda(x) \wedge x > 0 \supset \lambda(x - 1)) \vdash false.\) This, in turn, is equivalent to proving \((a \geq 0, \forall x (x = a \circ \lambda(x))) \wedge \forall x (\lambda(x) \wedge x \leq 0 \supset x = 0) \wedge \forall x (\lambda(x) \wedge x > 0 \supset \lambda(x - 1)) \vdash false\), a valid \(L\)-wff (and hence an axiom of \(\mathcal{D}\)), which can easily be proved in arithmetic using an induction axiom.

Another complete formal system in which total correctness can be
using rules D12-D15 (see remark after Theorem 6) to obtain only simple 
specifications (the validity of the deduction implies that the new symbols 
introduced at this stage will disappear in the process of deriving 
\( \sigma \)). Theorem 4 can now be applied. 

Our main result is

**Theorem 6** -

\[ \sigma_1, \ldots, \sigma_n \vdash \sigma \text{ iff } \sigma_1, \ldots, \sigma_n \vdash_D \sigma . \]

**Proof** - The only new feature here is the possibility of having call 
statements among the specification premises, with given declarations 
(implying that their "meaning" is fixed, and they can no longer be regarded 
as PSV's). Rule D16 is applied to all such procedures, effectively getting 
rid of the call's, and "trading" them in for new body-specifications. The 
situation is now precisely that described in the hypothesis of Theorem 5. 
Here too the validity of the original deduction implies that the new 
symbols \( \delta \) and \( \lambda \) (standing for the least fixpoints) will 
disappear in the derivation process. 

Note the decompose-collect-compose symmetry of the entire 
derivation process described in the above theorems:

1. "trade" call's for bodies  
2. decompose bodies and premises  
3. collect PSV information  
4. compose bodies  
5. "trade" bodies for call's  
6. compose conclusion.

As remarked above, step (2) shows up in a formal proof as the 
composition of the premises. This is a consequence of the deductive 
character of \( D \), the decomposed premises being "carried along" 
throughout the derivation and composed towards the end. However, we prefer 
to regard this step as "decompostion" because it is usually carried out 
first in a manner similar to subgoaling. A glance at the proof in the 
Appendix might help clarify this remark.

We remark here that restricting \( L \) to be first order can destroy the 
completeness, as shown in [9], a result which reminds one of (and in fact 
subsumes, and as such provides a new proof of) Wand's result [16]. This 
result and the rather obvious fact that if \( L \) is weak second order then it 
is expressive, should now be clarified by the complexity result for partial 
correctness appearing in [8].
proved is that introduced by Pratt[15] and proved complete in Harel et al[8]. Pratt's approach is to formulate a uniform induction principle explicitly in the system, in the form of a rule which is analogous to D5, and which composes a specification about the loop dual to partial correctness. In D, the dual to D5 is D15 (similarly for recursive calls), which merely "breaks up" the loop, providing all the information the loop specification carries with it, and leaves the rest to the logic of the underlying language.

(iii) (equivalence) Take programs

\( (P_1 \ldots P_n, \alpha), (T_1 \ldots T_m, \beta) \).

Their strong equivalence (see Manna[14]), can be proved in D by proving

\[ P_1 \ldots P_n, T_1 \ldots T_m, \delta(\alpha) \lambda \vdash \delta(\beta) \lambda \]

where \( \delta \) and \( \lambda \) are a new predicate symbols, and proving the dual (with \( \alpha \) and \( \beta \) exchanged). For example the reader might care to prove

\[
\begin{align*}
\text{P proc if p(x) then x=f(x); callP; callP else x=x fi end, T proc if p(x) then x=f(x); callT else x=x fi end, } \\
\delta(x) (\text{callP}(\lambda(x)) & \vdash \delta(x) \text{ (callT)(\lambda(x) )} \\
\end{align*}
\]

and its dual (a proof of this equivalence is given in the Appendix), or

\[
\delta(x, y) \text{ (while r(x) do x=f(x) od; while s(y) do y=g(y) od) } \lambda(x, y) \vdash \delta(x, y) \text{ (while r(x) vs(y) do if r(x) then x=f(x) else y=g(y) fi od) } \lambda(x, y)
\]

and its dual. (In both examples we write the elementary action with relation \( x' = f(x) \) as \( x = f(x) \).

(iv) (derived rules). Here we make use of a meta-theorem which states that if \( \sigma_1 \ldots \sigma_n \vdash \sigma \), then the following is a valid inference rule of D:

\[
\Gamma \vdash \sigma_1, \ldots, \Gamma \vdash \sigma_n
\]

\[
\Gamma \vdash \sigma
\]

For example proving

\[
pars, p \rightarrow r p q, \tau r \tau p, \tau r \rightarrow \tau p q, s(\alpha) t
\]

\[
\vdash p \text{ (while r do } \alpha \text{ od) } q
\]
in $D$, establishes the corresponding derived rule.

(v) (modular proofs). Take as a premiss anything previously established and prove the desired conclusion as a consequent. Sometimes it is possible to denote the established segment by a PSV and make the premises simple, this having the effect of shortening the proof and adding to its clarity.

(vi) (simplification and transformations) Using $D$, it is possible to validate general program transformations. Once a sufficient set of transformations has been established, this set can then be used to simplify, develop and synthesize correct programs. (See [2],[4],[6] and [13] for the use of such sets). Alternatively $D$ can be part of a program development system in which the user may create and validate his own transformations and apply them immediately to verified program segments.

Some simple examples of such transformations are

\[
\text{plif } r \quad \text{then } \alpha \quad \text{else } \alpha \quad \text{filq} \quad \vdash \quad \text{plq}
\]

\[
\text{pluhile } \neg p \quad \text{do } \alpha \quad \text{od; } \beta \quad \vdash \quad \text{pl} \beta \quad \text{q}
\]

\[
\text{plif } r \quad \text{then } \alpha \quad \text{else } \beta \quad \text{filq} \\
\vdash \quad \text{plif } \neg r \quad \text{then } \beta \quad \text{else } \alpha \quad \text{filq}
\]

Other examples are transformations for recursion removal (See[2]).

VI. Conclusion.

We have presented a complete system $D$, in which (besides providing for other important but somewhat less spectacular possibilities) equivalence and partial as well as total correctness of programs can be proved.

The notion of proof from assumptions can be regarded as a natural and important extension of the better known notion of proofs of program correctness using Hoare-like systems. If one chooses to take the view that Hoare's method essentially "cheats" by reducing the problem of proving a partial correctness specification to that of proving a formula of $L$, then
we might say that $D$ extends the "cheating" too, and reduces the problem of proving a deduction over partial correctness specifications to that of proving a deduction in $L$, and therefore requires a slightly stronger logical component than is needed in Hoare's system. The proofs of soundness and completeness of $D$ reduce to the traditional proofs of the same for Hoare's system when $D$ is stripped of its extra features. The relationship can be schematically seen by viewing $D$ in the following pictorial way:

\[
D = H \cup \{\lambda, v, 3, \forall\} \cup H^{-1}
\]

\[
\begin{align*}
\uparrow & \quad \uparrow & \quad \uparrow \\
\text{decompose} & \\
\downarrow & \quad \downarrow & \quad \downarrow \\
\text{collect} & \\
\downarrow & \\
\text{compose} & \\
\end{align*}
\]

Appendix.

We show how to prove the equivalence of the following two procedures:

\[
P \text{ proc if } p(x) \text{ then } x\leftarrow f(x); \text{ callP; callP else } x\leftarrow x \text{ fi end.}
\]

\[
T \text{ proc if } p(x) \text{ then } x\leftarrow f(x); \text{ callT else } x\leftarrow x \text{ fi end.}
\]

We make use of the following derived rule

\[
\Gamma \vdash p \lambda \alpha \lambda q, \quad \Gamma \vdash r [[p, q]] s
\]

\[
\Gamma \vdash r \lambda \alpha \lambda s
\]

Define $\Gamma$ as the set $\{\gamma \wedge \neg p \delta, \gamma \wedge p \{x \leftarrow fx\} \lambda, \lambda[[\gamma, \delta]] \mu, \mu[[\gamma, \delta]] \delta, \phi[[\gamma, \delta]] \psi\}$.

We refer to the declaration of $P$ as $P \text{ proc... end}$, and similarly for $T$.

1. $\Gamma, \text{ true}[[\gamma, \delta]] \neg p \vdash \gamma \wedge p \{x \leftarrow fx\} \lambda \quad \text{hyp.}$
2. $\Gamma, \text{ true}[[\gamma, \delta]] \neg p \vdash \lambda[[\gamma, \delta]] \mu \quad \text{hyp.}$
3. $\Gamma, \text{ true}[[\gamma, \delta]] \neg p \vdash \text{ true}[[\gamma, \delta]] \neg p \quad \text{hyp.}$
4. $\Gamma, \text{ true}[[\gamma, \delta]] \neg p \vdash \lambda[[\gamma, \delta]] \mu \neg p \quad \text{D2, D8(2,3)}$
5. $\Gamma, \text{ true}[[\gamma, \delta]] \neg p \vdash \mu[[\gamma, \delta]] \delta \quad \text{hyp.}$
(6) $\Gamma, \text{true} \{[\gamma, \delta]\} \vdash p \vdash \\
Y \lambda x,x'.(\mu x.(\gamma x) \delta x')) \delta (x')$  D12

(7) $\Gamma, \text{true} \{[\gamma, \delta]\} \vdash \mu \lambda y.\gamma \delta A1, L2[\text{with } x'=x]$ 

(8) $\Gamma, \text{true} \{[\gamma, \delta]\} \vdash \mu, \lambda y.\gamma \delta \vdash (\mu \lambda y.\gamma \delta) \delta$  hyp., L1

(9) $\Gamma, \text{true} \{[\gamma, \delta]\} \vdash \mu, \lambda y.\gamma \delta$ L2[we also use L3, and D8 with the empty program to create a conjunction of the hypotheses]

(10) $\Gamma, \text{true} \{[\gamma, \delta]\} \vdash \mu, \lambda y.\gamma \delta$  L3

(11) $\Gamma, \text{true} \{[\gamma, \delta]\} \vdash \mu, \lambda y.\gamma \delta$  L3, L2(7, 10)

(12) $\Gamma, \text{true} \{[\gamma, \delta]\} \vdash \mu, \lambda y.\gamma \delta$  L3(7, 10, D8)

(13) $\Gamma, \text{true} \{[\gamma, \delta]\} \vdash \lambda \{[\gamma, \delta]\} \delta$  D2(4, 12)

(14) $\Gamma, \text{true} \{[\gamma, \delta]\} \vdash \gamma \lambda p.(x \cdot fx); [\gamma, \delta]$  D3(1, 13)

(15) $\Gamma, \text{true} \{[\gamma, \delta]\} \vdash \gamma \lambda p.x \cdot x \delta$  hyp., D1

(16) $\Gamma, \text{true} \{[\gamma, \delta]\} \vdash \gamma \lambda p.x \cdot x \delta$ D4(14, 15)

(17) $\Gamma \hdash \text{proc... end, } \Gamma, \text{true} \{[\gamma, \delta]\} \vdash \gamma \{\text{call T}\} \delta$  D7

(18) $\Gamma \vdash \text{proc... end, } \Gamma, \text{true} \{[\gamma, \delta]\} \vdash \phi \{[\gamma, \delta]\} \psi$  hyp.

(19) $\Gamma \vdash \text{proc... end, } \Gamma, \text{true} \{[\gamma, \delta]\} \vdash \phi \{[\gamma, \delta]\} \psi$  DR

(20) $\Gamma \vdash \text{proc... end, } \gamma \{\text{if p then } x \cdot fx; [\gamma, \delta]; [\gamma, \delta] \text{ else } x \cdot x \phi \{[\gamma, \delta]\} \psi, \text{true} \{[\gamma, \delta]\} \vdash p \vdash \\
\phi \{\text{call T}\} \psi \vdash \text{hyp., D12, D13, D14}$

(21) $\Gamma \vdash \text{proc... end, } P \vdash \text{proc... end, } \phi \{\text{call P}\} \psi,$

true \{\text{call P}\} \vdash \phi \{\text{call T}\} \psi$  D16

(22) $\Gamma \vdash \text{proc... end, } P \vdash \text{proc... end, } \phi \{\text{call P}\} \psi$  L2(21, 22).

This establishes one direction. The other is very similar and uses

$\Gamma = \text{true} \{[\gamma, \delta]\} \vdash \gamma \lambda p.x \cdot x \delta \lambda$, 

$\lambda \{[\gamma, \delta]\} \delta \lambda, \phi \{[\gamma, \delta]\} \psi \lambda$.  We

now also need another fact about a call to T besides

true \{[\gamma, \delta]\} \vdash p. The new fact is needed in order to show that the second call to P leaves x unchanged. A suitable specification (which is proved as in line (22) above) is x = v \vdash p \{[\gamma, \delta]\} x = v, where v

is free. We omit the details of this direction.

Acknowledgements.

We wish to thank Nachum Dershowitz for suggestions following a detailed reading of a previous version of the paper. The first author benefited from many related discussions with Vaughan R. Pratt.
REFERENCES


