INDUCTIVE ASSERTION PATTERNS FOR RECURSIVE PROCEDURES

M.M. Fokkinga
Delft University of Technology
Dept. of Math.
132 Julianalaan
Delft - Netherlands

1. Introduction

Abstract. Hoare [4] has given the proof rule \[
\frac{p \land B(S)p}{p \text{ (while } B \text{ do } S)} p \land \neg B \quad (a) \] according to which we may infer the validity of the "correctness assertion" \( (b) \) from the "inductive assertion" \( (a) \).

We investigate the possibility of setting up such rules for recursive procedures, in which we only admit inductive assertions about elementary statements, and which will characterize the recursive procedures in the following senses:

(i) Let \( T \) be any program, then

the validity of the inductive assertions of the rule for \( P \) implies

the validity of the correctness assertion about \( T \), if and only if\n
\( P \) is semantically an extension of \( T \), i.e. \( T(x) = y \Rightarrow P(x) = y \).

(ii) Any correctness assertion about the procedure holds if and only if

it can be derived by means of the rule.

It will appear that the premiss of such a characterizing rule in general consists of an infinite set of inductive assertions (about elementary statements) and that a finite characterizing set exists if and only if

the procedure is "regular".

We treat the problem only for monadic (i.e. one variable only) recursive program schemes, as formalized in [2]. The interpretation of the schemes is in terms of relations rather than in terms of functions. So we write,

for an interpretation \( c \), rather \( (x,y) \in c(P) \) than \( c(P)(x) = y \).

This paper is a summary of Fokkinga, M.M. [2].
Origin of the work. In [2] De Bakker and Meertens gave a definition for the set of inductive assertions with which they achieved similar results. But they frequently used the sophisticated Scott's Induction Rule in their argumentation and they did not include the possibility of a finite pattern, nor did they formulate characterization (ii). Originally we aimed to give a more direct definition and an argumentation without using Scott's Induction Rule and in addition we wanted to analyze their introductory "attempts that failed".

2. Sketch of the intuitive idea

Let $A_1; A_2; \ldots; A_n$ be a statement scheme consisting only of elementary statement symbols and the sequent symbol $;$. Then it is not difficult to give a set $\mathcal{A}$ of inductive assertions so that "infer from $\mathcal{A}$ the correctness assertion $p_n(A_1; \ldots; A_n)_{\text{ex}}"$ is a rule characterizing the scheme $A_1; \ldots; A_n$.

Indeed, let $\mathcal{A}$ consist of $p_{i_{\text{in}}} \Rightarrow p_1$ and $p_1(A_i)_{p_{i_{\text{ex}}}} (i = 1, \ldots, n-1)$ and $p_1(A_n)_{p_{\text{ex}}}$. Then, (a) for any interpretation the validity of $\mathcal{A}$ clearly implies the validity of the correctness assertion. Moreover, (b) if $T$ is some scheme and for any interpretation the validity of $\mathcal{A}$ implies the validity of $p_{i_{\text{in}}}(T)_{p_{\text{ex}}}$, then we can prove that for every interpretation $c$ $A_1; \ldots; A_n$ is an extension of $T$.

(proof: let $(x_0, y_0) \in \text{c}(T)$, then we have to show $(x_0, y_0) \in \text{c}(A_1; \ldots; A_n)$.

Consider the hypothesis under a particular interpretation $c'$ which is obtained from $c$ by giving the predicate symbols

$$p_{i_{\text{in}}} \equiv p_0; p_1; \ldots; p_n; p_{n+1} \equiv p_{\text{ex}}$$

The following meaning:

$c'(p_i)$ holds for $x \overset{\text{def}}{\leftarrow} (x_0, x) \in \text{c}(A_1; \ldots; A_{i-1})$.

This is possible because the $p_i$ do not occur in $T$ and $A_1; \ldots; A_n$.

Then the assertions in $\mathcal{A}$ are valid under $c'$, hence also $p_{i_{\text{in}}}(T)_{p_{\text{ex}}}$ is valid. Now because $c'(p_{i_{\text{in}}})$ holds trivially for $x_0$, we may conclude that $c'(p_{\text{ex}})$ holds for $y_0$ (recall that $(x_0, y_0) \in \text{c}(T)$).

By definition, this means

$$(x_0, y_0) \in \text{c}(A_1; \ldots; A_n). \text{ Q.E.D.}$$

Hence (c), from (a) and (b) it follows that the rule characterizes the scheme $A_1; \ldots; A_n$ in the sense (i) given above.
This approach seems applicable even with schemes of more complex structure than the simple sequence of elementary statement symbols. For instance, an evaluation for some input of a procedure symbol P ultimately has to result in a sequence of executions of elementary statement symbols $A_1; \ldots; A_n$. Call such a sequence an evaluation sequence. When we define for each evaluation sequence a set of inductive assertions just as above, and when we let the set $\mathcal{A}$ of inductive assertions for P be the union of them, then $\mathcal{A}$ characterizes the scheme consisting of procedure symbol P.

The approach just described is worked out in the sections 4, 5 and 6.

3. Preliminaries on program schemes

The program schemes $(P, T, \ldots)$ which we consider are formulated in [2]. They can be defined as follows.

A program scheme $P$ is an ordered pair $P \equiv <D, P_0>$, where $D = \{P_0 \leftarrow S_0, \ldots, P_n \leftarrow S_n\}$ is a declaration scheme and the bodies $S_0, \ldots, S_n$ are statement schemes, which are inductively constructed from

- elementary statement symbols ($A$, with indices)
- procedure symbols ($P$, with indices)
- constant symbols (identity symbol $E$, emptiness symbol $\emptyset$)

by means of binary compositions with pairs of parentheses and

- the alternation symbol $\vee$
- the sequentiation symbol $\cup$

An interpretation $c(P)$ for a program scheme $P \equiv <D, P_0>$ is the interpretation $c(P_0)$ of the statement scheme $P_0$ under an interpretation $c$ with respect to the declaration scheme $D$.

An interpretation $c$ consists of the choice of a set, the domain $\text{Dom}(c)$, and the choice of a binary relation $c(A)$ on $\text{Dom}(c)$ for each relation symbol $A$.

The constant symbols have a fixed interpretation under all interpretations $c$: $c(E)$ is the identity relation $\{(x, x) \mid x \in \text{Dom}(c)\}$ and $c(\emptyset)$ is the empty relation on $\text{Dom}(c)$.

The interpretation $c(S)$ of a statement scheme $S$ is a binary relation on $\text{Dom}(c)$ defined by the notion of computation sequence, which precisely reflects the copy rule for procedure calls.
Predicate symbols \((p, B, \forall B, \ldots)\) are elementary statement symbols, which are interpreted as subrelations of \(c(E)\); the negation sign is interpreted as \(c(\forall B) = \{(x, x) \mid x \in \text{Dom}(c) \land (x, x) \notin B\}\).

Define an operation ";" for relations to be the concatenation, \((x, y) \in R_1; R_2 \iff \exists z: (x, z) \in R_1 \land (z, y) \in R_2\) and an operation "\(v\)" to be the union of relations. Then \(\;\) and \(\cup\) are associative and for schemes \(S_1\) and \(S_2\) we have, for all interpretations \(c\), \(c(S_1; S_2) = c(S_1)\) \(\land\) \(c(S_2)\) and \(c(S_1 \cup S_2) = c(S_1) \cup c(S_2)\).

Let \(\mathcal{A}\) and \(\mathcal{C}\) stand for collections of assertions about interpretations of schemes. The statement that "the validity under interpretation \(c\) of the assertions in \(\mathcal{A}\) implies the validity under \(c\) of the assertions in \(\mathcal{C}\)" is symbolized in the formula \(\mathcal{A} \models_c \mathcal{C}\). In particular \(\models_c \mathcal{C}\) is the statement that the assertions in \(\mathcal{C}\) are valid under \(c\). We abbreviate "for all interpretations \(c\) \(\mathcal{A} \models_c \mathcal{C}\)" by \(\mathcal{A} \models \mathcal{C}\).

We use as assertions about interpretations of schemes only the set-theoretic inclusion, symbolized by the connective \(\subseteq\). The connective \(\leq\) is used for the equality on the domain of the interpretation. Thus \(S_1 \leq S_2\) is short for \((S_1 \subseteq S_2)\land(S_2 \subseteq S_1)\). The connectives \(\limp\) and \(\wedge\) are used to express the (stronger) syntactical relations with respect to formal languages and pure formal objects. A set \(\mathcal{A}\) of inductive assertions is called an inductive assertion pattern.

**Example.** The program scheme determined by \(P_0 \leq B; A; P_0 \cup \forall B\) is the procedural form of the while statement. Hoare's rule now becomes \(p; B; A \leq A; p \models p; P_0 \leq P_0; p; \forall B\).

4. **Program schemes and grammars**

We employ context-free grammars as a tool for describing the evaluation of recursive procedures. Let \(P\) be an program scheme. We associate with \(P\) a c.f. grammar \(G\)-of-\(P\):

- the nonterminals are and correspond to procedure symbols of \(P\),
- the terminals are and correspond to the elem. stat. symbols,
- the derivations rules are and correspond to procedure declarations,
- the alternatives in the rules correspond to because such symbols \(v\) in the schemes occurring in \(P\).
Due to the correspondences and the fact that the symbols of $G$-of-$P$ are symbols of the scheme $P$, it makes sense to speak of interpretations of syntactical objects. In particular, we define for a language $L$ $c(L) = \cup \{ c(\tau) \mid \tau \in L \}$.

**THM** $P = L(G$-of-$P)$

**proof**: by induction we can prove for arbitrary interpretation $c$ and $x$ and $y$ in $\text{Dom}(c)$:

- there exists a computation sequence $x(P_0) \ldots y$, i.e. $(x,y) \in c(P)$, if and only if
- there exists a left most derivation $P_0 \Rightarrow \tau$ in $G$, with $(x,y) \in c(\tau)$, i.e. $(x,y) \in c(G)$.

5. *Inductive assertion patterns and tied complete derivation tries*

A generalization concerning "tying the trees" is treated within the square brackets $[and]$. A generalization concerning "completeness" is treated within the Kleene brackets $\{and\}$. The definition without the additions between these brackets, precisely covers the usual concept of derivation tree of formal language theory.

**Def. 1** A [tied] {complete} derivation tree $\mathcal B$ for a nonterminal or terminal symbol $S$ of a context-free grammar, where $S$ is called the root symbol of $\mathcal B$, is a diagram consisting of

- a [not necessarily] new occurrence of the symbol $S$, where the occurrence is called the root of $\mathcal B$,

- with in case $S$ is a terminal symbol nothing, but in case $S$ is a nonterminal symbol a by this occurrence of $S$ uniquely determined sequence (resp. uniquely determined collection of uniquely determined sequences) of [tied]{complete} derivation trees, that is to say

  for a derivation rule with lefthand side $S$ one such tree for consecutively, say from left to right, every symbol in the righthand side of the derivation rule (and just one such sequence for every derivation rule with lefthand side $S$)

(N.B. uniqueness does not imply that sequences are not allowed to coincide).

A node of the tree is an occurrence of a symbol of the tree.
We picture this definition as $B$: $S$ in case $S$ is a terminal symbol, and $\overrightarrow{B}$ otherwise, where $\overrightarrow{B}_1$ is the root of $\overrightarrow{B}_1$ or a reference "→" to it.

Example. Let a grammar be given by $Z \rightarrow aZb$, $Z \rightarrow c$. Then $B'$ and $B''$ are some (not-tied) der. trees for $Z$, and $B$ is the (not-tied) complete derivation tree for $Z$, and $\overrightarrow{B}_1$ and $\overrightarrow{B}_2$ are some tied compl. der. trees for $Z$: 

Def. 2 For a [tied] {complete} derivation tree $B$:

A (left most/right most) direct subtree is every (L/R most) element from the sequence {s} as mentioned in 5.2.

A (left most/right most) subtree is the tree $B$ itself, and every (L/R most) subtree of any (L/R most) direct subtree of $B$.

(Directly) successive are two direct subtrees $U$ and $V$ if $U$ and $V$ are elements of {one of} the sequence {s} mentioned in 5.2. and $V$ is some (resp. the) sequence element following $U$.

(Directly) successive are two subtrees $X$ and $Y$ if there are two direct subtrees $U$ and $V$ of a subtree of $B$, such that $U$ and $V$ are (directly) successive and $X$ is (right most) subtree of $U$ and $Y$ is (left most) subtree of $V$.

(Directly) successive are two nodes if the subtrees of which they are the roots, are (directly) successive.

Example. For the previously pictured $B'$, $B''$, $B$ the relations are obvious. For $B_1$ we can state, among others, that:

a upper is L-most subtree (node) and b upper is R-most subtree (node),

a lower is dir. successor of a upper and b upper is dir. successor of b lower

c upper is L/R-most subtree (node) and c lower is dir. successor of a upper.
and due to tying the tree:
the subtree determined by the reference \( \circ \) in the bottom line coincides
to the whole tree \( B_1 \), hence

\[ a \text{ upper is dir. successor of } a \text{ lower and } b \text{ lower is dir. successor of } b \text{ upper,} \]

\[ c \text{ upper is dir. successor of } a \text{ lower and } b \text{ lower is dir. successor of } c \text{ upper.} \]

Def. 3 A (direct) production is a sequence of directly successive (direct) subtrees
\( B_1; \ldots; B_n \) of \( A \), such that \( B_1 \) and \( B_n \) are resp. a left most and a right
most subtree of \( A \). A terminal subtree (node) is a subtree which is an
occurrence of a terminal symbol. A terminal production is a production
consisting of terminal subtrees. A sentential form (resp. sentence)
is the sequence of root symbols of a (terminal) production. The language
\( L(B) \) of \( B \) is the set of sentences of \( B \). We denote a node which is
an occurrence of the symbol \( A \) by \( \hat{A} \).

Example. \( L(B') = \emptyset \), \( L(B'') = \{ \text{acb} \} \), \( L(B) = \{ a^n c b^n \mid n \geq 0 \} \),
\( L(B_1) = \{ a^m c b^n \mid m, n \geq 0 \land n \equiv m \mod{2} \} \), \( L(B_2) = \{ a^m c b^n \mid m, n \geq 0 \} \).

From the definitions follows immediately the

THM The language generated by a c.f. grammar equals the language of the not-tied
complete derivation tree for (the sentence symbol of) that grammar.

The definition
of the Inductive Assertion Pattern \( \mathcal{A} \) based upon a [tied] complete derivation
tree \( B \) and with respect to correctness predicate symbols \( p_{\text{in}} \) and \( p_{\text{ex}} \) reads:

Let \( \{ p_i \}_{i \in I} \) be a collection of new predicate symbols which are
bijectively associated with the terminal subtrees (nodes) of the tree \( B \).

Then \( \mathcal{A} \) consists of the following assertions
- for directly successive terminal subtrees (nodes) \( \hat{A} \) and \( \hat{A}' \) (with associated
  predicated symbols \( p \) and \( p' \)) the inclusion \( p; A \subseteq A' ; p' \)
- for left most and right most terminal subtrees (nodes) \( \hat{A} \) and \( \hat{A}' \) (with predicate
  symbols \( p \) and \( p' \)) the inclusions \( p_{\text{in}} \subseteq p \) resp. \( p'; A' \subseteq A' ; p_{\text{ex}} \).
6. The characterization theorem

In the sequel
we let \( \mathcal{A} \) be an ind. ass. pattern based upon some tied complete der. tree \( \mathcal{B} \)
and we let \( P \) be any program scheme with associated context-free grammar \( G \)
and \( \mathcal{E}(..) \) is an abbreviation of \( p_{in}; ..; p_{ex} \).
Note that henceforth \( \mathcal{B} \) need not be related to \( G \); we want to investigate their
characterizing power and therefore they must be unrelated entities.

The essential proof - without employing Scott's Induction Rule - follows in the Main lemma

(i) \( \mathcal{A} \models \mathcal{E}(L(\mathcal{B})) \)

(ii) for any scheme \( T \), \( \mathcal{A} \models \mathcal{E}(T) \) implies \( \vdash T \in L(\mathcal{B}) \)

(iii) \( \mathcal{E}(L(\mathcal{B})) \models \mathcal{A} \).

(*: provided the pred. sym. \( p_i \) in \( \mathcal{A} \) are conveniently interpreted).

Proof

(i). Let \( c \) be any interpretation and let \( T \equiv A_1; \ldots; A_n \in L(\mathcal{B}) \).
Then there are directly successive terminal subtrees (nodes) \( A_1, \ldots, A_n \)
in \( \mathcal{B} \), of which \( A_1 \) and \( A_n \) are left most and right most ones.
Consequently the inclusions \( P_{in} \subseteq P_1 \) and \( p_i; A_i \in A_i; p_{i+1} \) for
\( i = 1, \ldots, n-1 \) and \( p_n; A_n \in A_n; p_{ex} \) belong to \( \mathcal{A} \). From the validity of \( \mathcal{A} \)
under interpretation \( c \) there follows consecutively for \( i = 0, 1, \ldots, n-1 \)
the validity under \( c \) of

\[
P_{in}; A_i; \ldots; A_i; p_{i+1} \subseteq A_1; \ldots; A_i; p_i+1 \quad \text{so}
\]

\[
P_{in}; A_i; \ldots; A_i; i+1; A_{i+1} \subseteq A_1; \ldots; A_i; p_{i+1}; A_{i+1} \quad \text{so according to} \quad p_{i+1}; A_{i+1} \subseteq A_1; p_{i+2}
\]

\[
P_{in}; A_i; \ldots; A_i; i+1 \subseteq A_1; \ldots; A_i; A_{i+1}; p_{i+2}
\]

where for \( i = n+1 \) \( p_{i+2} \equiv p_{ex} \). So

\( \mathcal{A} \models P_{in}; T \subseteq T; p_{ex} \) for any \( T \in L(\mathcal{B}) \), so also

\( \mathcal{A} \models U(P_{in}; T) \subseteq U(P_{in}; T) \) the union taken over all \( T \in L(\mathcal{B}) \),

\( \mathcal{A} \models P_{in}; U(T) \subseteq p_{in}; U(T) \) Q.E.D.

(ii). Let \( c \) be any interpretation, we show \( \vdash_c T \in L(\mathcal{B}) \). Thus let \( x_0 \) and
\( y_0 \) be arbitrary elements in \( \text{Dom}(c) \) with \( (x_0, y_0) \in c(T) \), then we have to show
\( (x_0, y_0) \in c(L(\mathcal{B})) \).
Consider the premiss under the particular interpretation \( c' \), obtained from \( c \) merely by changing or defining the interpretations of the predicate symbols \( p_{in} \), \( p_{ex} \) and \( \{ p_i \}_{i \in I} \). Because they do not occur in \( T \) and \( L(\mathcal{B}) \), we have \( c(T) = c'(T) \) and \( c(L(\mathcal{B})) = c'(L(\mathcal{B})) \).

Informally, we give \( p_{in}, p_{ex}, p_i \) \((i \in I)\) the meaning that holds true for those arguments \( x \) which result from input \( x_o \) and a computation by the successive \( A_1, \ldots, A_{i-1} \) (which occur as an initial segment of a sentence in the tree \( \mathcal{B} \)) up to, but not including, the symbol \( A_i \) with which the predicate symbol is associated. Herein we consider \( p_{in} \) and \( p_{ex} \) to be associated with an imaginary begin and end marker of sentences of \( \mathcal{B} \).

Formally the definition reads:

\[
(x, x) \in c'(p) \quad \text{def} \quad \exists \hat{A}_1, \ldots, \hat{A}_i \text{ such that } \hat{A}_i \text{ is the leftmost one and } (x, x) \in c(A_1; \ldots; A_{i-1}) \text{ and } p \text{ is associated with } \hat{A}_1.
\]

\[
(x, x) \in c'(p_{in}) \quad \text{def} \quad x = x_o
\]

\[
(x, x) \in c'(p_{ex}) \quad \text{def} \quad (x_o, x) \in c(A_1; \ldots; A_n) \text{ for some terminal production } \hat{A}_1; \ldots; \hat{A}_n \text{ of } \mathcal{B}.
\]

It is now easy to verify the validity of \( \mathcal{P} \) under \( c' \):

- if \( p; A \subseteq A' \); \( p' \) belongs to \( \mathcal{A}' \), then \( p \) and \( p' \) are associated with directly successive terminal nodes \( \hat{A}, \hat{A}' \). So from \( (x, y) \in c'(p; A) \) it follows that \( (x, y) \in c'(A) \) and \( (x_o, x) \in c(A_1; \ldots; A_{i-1}) \) for some sequence of directly successive terminal nodes \( \hat{A}_1, \ldots, \hat{A}_i, \hat{A}_i \equiv \hat{A} \).

So we have \( (x, y) \in c'(A) \) and \( (x_o, y) \in c(A_1; \ldots; A_{i-1}); c(A_i) = c(A_1; \ldots; A_i) \), i.e. \( (x, y) \in c'(A; p') \).

- if \( p_{in} \subseteq p \) belongs to \( \mathcal{A} \), then trivial

- if \( p; A \subseteq A_i p_{ex} \) is in \( \mathcal{A} \), then analogously to the first case.

So according to the premiss of the lemma \( p_{in}; T \subseteq T; p_{ex} \) holds under \( c' \).

We have assumed \( (x_o, y_o) \in c(T) \) so that \( (x_o, y_o) \in c'(p_{in}; T) \) holds, too. So with the inclusion just derived it follows that \( (x_o, y_o) \in c'(T; p_{ex}) \) and in particular \( (y_o, y_o) \in c(p_{ex}) \). By definition this means \( (x_o, y_o) \in c(T) \) for some sentence \( \tau \in L(\mathcal{B}) \), hence \( (x_o, y_o) \in c(L(\mathcal{B})) \). Q.E.D.
(iii). The convenient interpretation of the pred. symbols in $\mathcal{A}$ is the following: let $c$ be given for all symbols except for the $p_i$ ($i \in I$), then $(x,x) \in c(p) \iff$ there is some $x_0$ in $\text{Dom}(c)$ such that $(x_0,x) \in c(p_{in}; A_1; \ldots; A_{i-1})$ for an initial segment $A_1; \ldots; A_i$ of some sentence of $\mathcal{L}$ where $p$ is associated with $A_i$.

Now suppose $\mathcal{E}(L(\mathcal{B}))$ is valid under some $c$, then it is easy to verify the validity of the assertions in $\mathcal{A}$ under $c$ with the convenient interpretation for the $p_i$ ($i \in I$):

for $p_{in} \in p$ and $p; A \in A; p'$ in $\mathcal{A}$ it is straightforward, and

for $p; A \subseteq A; p_{ex}$ we argue as follows: let $(x,y) \in c(p; A)$ then by

definition $(x_0,x) \in c(p_{in}; A_1; \ldots; A_{n-1})$ and $(x,y) \in c(A_n)$ where $A_n \equiv A$

and $p$ is associated with $A$, so $(x_0,y) \in c(p_{in}; A_1; \ldots; A_n)$ for some sentence of $L(\mathcal{B})$, hence by the assumption of the validity of $\mathcal{E}(L(\mathcal{B}))$ under $c$ we get

$(x_0,y) \in c(A_1; \ldots; A_n; p_{ex})$. So both $(x,y) \in c(A)$ and $(y,y) \in c(p_{ex})$ hold, hence $(x,y) \in c(A; p_{ex})$. Q.E.D.

Lemma

(i) $\mathcal{A} \models \mathcal{B}(P)$ iff $L(\mathcal{B}) \supseteq L(G)$

(ii) for any scheme $\mathcal{T}$, $\mathcal{A} \models \mathcal{B}(T)$ implies $\models \mathcal{T} \subseteq P$ iff $L(\mathcal{B}) \subseteq L(G)$

(iii) $\mathcal{B}(P) \models \mathcal{A}$ iff $L(\mathcal{B}) \subseteq L(G)$

(*: provided the $p_i$ in $\mathcal{A}$ are conveniently interpreted, in case we read the equivalence from right to left).

Proof. Apply the Main lemma and use $\models P = L(G)$ and the easily provable fact

$L_1 \subseteq L_2$ iff $\models L_1 \subseteq L_2$ for languages $L_1$ and $L_2$. Use in (iii)$\Rightarrow$ an interpretation $c$ such that the validity under $c$ of $\mathcal{B}(L(\mathcal{B}))$ is equivalent with $L(\mathcal{B}) \subseteq L(G)$.

Characterization theorem

Any of the following three characterizations holds if and only if $L(\mathcal{B}) \equiv L(G)$:

(i) for any scheme $\mathcal{T}$

$\mathcal{A} \models \mathcal{B}(T)$ if and only if $P$ is an extension of $T$, i.e. $\models T \subseteq P$

(ii) for any fixed point $T$ of the scheme $P$

$\mathcal{A} \models \mathcal{B}(T)$ if and only if $T$ is the minimal fixed point of $P$, i.e. $\models T = P$

(iii) $\mathcal{A} \models \mathcal{B}(P)$ is a "complete" proofrule for assertions about $P$.

Proof. By dull manipulations of the previous results and noting that (ii) is a consequence of (i).
Conclusion In order that the Inductive Assertion Pattern $\mathcal{A}$ based upon some tree $\mathcal{B}$ - characterizes the scheme $P$ - with associated c.f. grammar $G$ -, we have to choose $\mathcal{B}$ such that $L(\mathcal{B}) \equiv L(G)$. This equality is easily achieved when we choose $\mathcal{B}$ as the not-tied complete derivation tree for (the sentence symbol of) the grammar $G$. See the theorem in section 5. But then $\mathcal{A}$ is infinite as soon as $P$ contains a recursive procedure. This is unsatisfactory, the more so as the while statement can be considered as a recursive procedure and the only assertion of Hoare’s proof rule characterizes the while statement! The question arises whether we cannot define a more economic pattern $\mathcal{A}$.

We can achieve finiteness by tying the underlying tree $\mathcal{B}$ so that we get a finite diagram. This possibility is achieved by the additions between the square brackets [and] in the definitions of section 5.

Finiteness Theorem

$\mathcal{B}$, hence $\mathcal{A}$, can be chosen finite with $L(\mathcal{B}) \equiv L(G)$ if and only if $L(G)$ is regular (and then we say $P$ is regular).

Proof. If $\mathcal{B}$ is finite (in the number of terminal nodes) then we can construct a finite automaton by considering $\mathcal{B}$ as such and which accepts $L(\mathcal{B})$ due to the control mechanism as induced by the first clause of def. 3 of section 5.

Conversely, for every grammar $G'$ in regular form we can construct a finite tree $\mathcal{B}$ with $L(\mathcal{B}) \equiv L(G')$, by tying whenever possible for nonterminal nodes.

We have given a very general definition of tying trees. In the proof above we used a particular case: very straightforward tying whenever possible (for nonterminals). By choosing a convenient way of tying, we can get all introductory patterns of [2, section 4.1.] which appeared to fail. There some patterns were proposed as characterizing a recursive procedure $P$, but whereas $\mathcal{A} \vdash E(P)$ was true, the implication "$\mathcal{A} \vdash E(T)$ implies $T \subseteq P$" was not. The cause is now clear: the language $L(\mathcal{B})$ of the tied tree on which $\mathcal{A}$ was based, did not equal the language $L(G$-of-P) generated by the grammar $G$-of-P associated with the scheme $P$.

Counterexample theorem

A program scheme $T$ is a counterexample of "$\mathcal{A} \vdash E(T)$ implies $T \subseteq P$" if and only if the grammar $G$-of-$T$ associated with $T$ satisfies the following requirements: (i) $L(G$-of-$T) \subseteq L(\mathcal{B})$ (ii) $L(G$-of-$T) \not\subseteq L(G$-of-$P$)

Proof By manipulations of the previous results.
7. Example/Application: the while statement characterization

The while statement $W \equiv \text{while } B \text{ do } A$ can be considered as recursively defined by $W \leftarrow B; A; W \cup \neg B$.

The c.f. grammar associated to $W$ is given by

$W \rightarrow (B; A; W \cup \neg B) \ (B; A; W \cup \neg B) + B; A; W \ (B; A; W \cup \neg B) + B$

An equivalent grammar in regular form is

$W \rightarrow B; A; W \rightarrow \neg B$

The tree $\mathcal{B}$ constructed according the Finiteness Thm is:

The pattern $\mathcal{A}$ based upon $\mathcal{B}$, is according to the definition:

$\mathcal{A} : p_{in} \leq p_B \quad p_B;B \leq B; p_A \quad p_A;A \leq A; p_B \quad p_B;A \leq A; p_\neg B \quad p_{in} \leq p_\neg B \quad p_\neg B;\neg B \leq \neg B; p_{ex} \quad$ 

By contracting the 2nd + 3rd and the 2nd + 4th inclusion we can eliminate $p_A$.

Thus we obtain the equivalent *) pattern

$\mathcal{A}' : p_{in} \leq p_B \quad p_B;B;A \leq B; A; p_B \quad p_B;B;A \leq B; A; p_\neg B \quad p_{in} \leq p_\neg B \quad p_\neg B;\neg B \leq \neg B; p_{ex} \quad$

Because $B$ and $\neg B$ must be interpreted as "predicate relations", it is easy to verify by elementswise consideration that $\mathcal{A}'$ is equivalent *) to

$\mathcal{A}'' : p_{in} \leq p_B \quad p_B;B;A \leq A; p_B \quad p_B;A \leq A; p_\neg B \quad p_{in} \leq p_\neg B \quad p_\neg B;\neg B \leq p_{ex} \quad$

And finally, by suitable substitutions for the predicate symbols $p_B$ and $p_\neg B$, respectively for the new $p_1$, $\mathcal{A}''$ is easily demonstrated to be equivalent *) to

$\mathcal{A}''' : p_{in} \leq p_1 \quad p_1;B;A \leq A; p_1 \quad p_1;\neg B \leq p_{ex} \quad$

According to the theorems we have that

(i) for any scheme $T$

$\mathcal{A}''' \models p_{in};T \leq T; p_{ex}$ if and only if $\models T \leq W$

(ii) for any scheme $T$ which satisfies $\models T = B; A; T \cup \neg B$

$\mathcal{A}''' \models p_{in};T \leq T; p_{ex}$ if and only if $\models T = W$

(iii) both $\mathcal{A}''' \models p_{in};W \leq W; p_{ex} \quad$ and $p_{in};W \leq W; p_{ex} \models \mathcal{A}'''$

*) in the sense $\mathcal{A}_1 \models \mathcal{A}_2$ and $\mathcal{A}_2 \models \mathcal{A}_1$ (*: provided the predicate symbols in the righthand side are conveniently interpreted).
References

[1] de Bakker, J.W.,
   Recursive procedures

   Simple recursive program schemes and inductive assertions

[3] de Bakker, J.W., and de Roever, W.P.,
   A calculus for recursive program schemes
   Proc. IRIA Symp. on Automata, Form. Lang., Programming
   North Holland, Amsterdam (ed. Nivat)

   An axiomatic basis for computer programming

[5] Fokkinga, M.M.,
   Inductive Assertion Patterns for Recursive Procedures

Recently there has appeared:

   On the completeness of the inductive assertion method
   which is a generalisation of [2] and includes our results, however proved in
   a different way.