INDUCTIVE ASSERTION PATTERNS FOR RECURSIVE PROCEDURES

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

Contents

A shortened version is intended for publication in Proceedings of Symposium on Programming, Paris 9-11 April 1974.

1. Introduction

1.1. Abstract

Hoare [IV] has given the proof rule $\dfrac{P \wedge B\{S\}P}{P\ \{\underline{\text{while}}\ B\ \underline{\text{do}}\ S\}\ P \wedge \neg B}\ \begin{matrix}(a)\\(b)\end{matrix}$ according to which we may infer the validity of the "correctness assertion" (b) from the "inductive assertion" (a). The rule even characterizes the while statement in the following sense:

(i) Let T be any program satisfying T = if B then (S;T). Then the validity of P ∧ B{S}P implies the validity of P{T}P ∧ ¬B, } for all specifications of P
if and only if T = while B do S, (Scott, see[I]).

N.B. Here and in the sequel the equality of programs concerns the input-output behaviour.

The rule also characterizes the while statement in the following sense:

(ii) any correctness assertion about the while statement holds, if and only if it can be derived by means of the rule (and some additional rules, such as: P ⇒ R, R{T} S, S ⇒ Q implies P{T} Q).

We investigate the possibility of setting up such characterizing rules, in which we only admit inductive assertions about *elementary* statements, for recursive procedures. It will appear that the premiss of such a characterizing rule in general consists of an infinite set of inductive assertions (about *elemantary* statements) and that a finite characterizing set exists if and only if the procedure is "regular". We will derive the analogue for recursive procedures of (ii) and the following characterization:

(iii) 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, } for all specifications of the ind. ass. and the correctness assertion
if and only if P is semantically an extension of T, i.e.
$T(x) = y \to P(x) = y$.

From (iii) the analogue for recursive procedures of (i) is easily obtained, i.e.

(i') Let T be any fixed point of the recursive program P, Then
the validity of the inductive assertions of the rule for P
implies the validity of the correctness assertion about T, } for all specifications of the pred. symbols in the ind. ass's and in the correctness ass.:
if and only if T is the minimal fixed point, i.e. $T = P$.

1.2. Origin of the work

In [I] 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".

1.3. Restriction

We treat the problem only for monadic (i.e. one variable only) recursive program schemes,as formalized in [I]. 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 $c(P)(x) = y$.

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 sequention symbol; .
Then it is not difficult to give a set $\mathcal{A}$ of inductive assertions so that "infer from $\mathcal{A}$ the correctness assertion $P_{in}\ \{A_1;\ldots;A_n\}P_{ex}$" is a rule characterizing the scheme $A_1;\ldots;A_n$.
Indeed, let $\mathcal{A}$ consist of $P_{in} \Rightarrow P_1$ and $P_i\{A_i\}P_{i+1}$ $(i= 1,\ldots,n-1)$ and $P_n\ \{A_n\}\ P_{ex}$. 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 $P_{in}\ \{T\}P_{ex}$, then we can prove that for every interpretation c $A_1;\ldots;A_n$ is an extension of T.

(<u>proof</u>: let $(x_0,y_0) \in c(T)$, then we have to show $(x_0,y_0) \in c(A_1;\ldots;A_n)$. consider the hypothesis under a particular interpretation c' which is obtained from b by giving the predicate symbols $P_{in} \equiv P_0,\ P_1,\ldots,\ P_n,\ P_{n+1} \equiv P_{ex}$ the following meaning:
$c'(P_i)$ holds for $x \leftrightarrow_{def} (x_0,x) \in c(A_1;\ldots;A_{i-1})$.
This is possible because the P_i do not occurr in T and $A_1,\ldots,A_n$.
Then the assertions in $\mathcal{A}$ are valid under c', hence also $P_{in}\{T\}\ P_{ex}$ is valid. Now because $c'(P_{in})$ holds trivially for x_0, we may conclude that $c'(P_{ex})$ holds for y_0 (recall that $(x_0,y_0) \in c(T)$). By definition, this means $(x_0,y_0) \in c(A_1;\ldots;A_n)$. 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 (iii) 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 uninion 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.

In section 4:

In order to get a formal definition for this approach, we employ context-free grammars as a tool for describing the evaluation of – recursive – procedures. Let P be a program scheme. We associate with P a c.f. grammar G:

- the nonterminals of G are and correspond to the procedure symbols of P
- the terminal symbols are and correspond to the elementary statement symbols
- the derivation rules are and correspond to the procedure declarations
- the alternatives in the rules correspond to the choice constructs (e.g. if then else) in the schemes.

Then the sentences of G are precisely the evaluation sequences of P.

In section 5:

For the sake of a definition as explicit as possible, we define the concept of a *complete derivation tree* as an extension of the well known concept of derivation tree from formal language theory. In a complete derivation tree there descend from each nonterminal symbol as many subtrees as there are production rules for that nonterminal in the given grammar. The sentence of a derivation tree is the sequence of terminal symbols next to each other, to begin with the left most one and to end with the right most one. We extend the relation "next to each other" (and will call it "directly successive") so that the complete derivation tree produces a collection of sentences rather than one sentence, that is to say it produces the complete language of the given grammar and hence the complete set of evaluation sentences of the program with which the grammar is associated.

The definition of the set $\mathcal{A}$ of inductive assertions with respect to the correctness predicate symbols P_{in} and P_{ex} now becomes:

> associate with each terminal symbol A in the complete derivation tree $\mathcal{B}$ for the grammar associated with the program scheme P a new predicate symbol p. Then $\mathcal{A}$ consists of
>
> - for terminal symbols A and A' (with associated predicate symbols p and p') which are next to each other, the assertion p{A}p'
> - for left most and right most terminal symbols A and A' (with associated predicate symbols p and p') the assertions $P_{in} \Rightarrow p$ and $p'\{A'\}P_{ex}$.

In section 6:
Indeed we can now derive the required results with this definition. Essentially this is done in the Main Lemma -- quite similar to the proof given above -. But the set $\mathcal{A}$ is infinite as soon as the program scheme 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.
Therefore we make the definitions more general so that they include finite sets, too. We only change the definition of the trees: now we allow trees to be *tied*, that is the branches of a node need not necessarily descend to nodes beneath, but may be tied to nodes above and beside. The relation "next to each other" is extended so that in any case the sentences of the not-tied complete derivation tree will be produced by every tied complete derivation tree based upon the same grammar. But in general the language of a tied complete derivation tree includes the language of the not-tied one properly.
Furthermore, we base the definition of the set $\mathcal{A}$ of inductive assertions on an arbitrary tied complete derivation tree $\mathcal{B}$ and not necessarily on a tree for the grammar associated with the program scheme *P*.
Now under some conditions all proofs and theorems still hold. Therefore we treat these two generalisations immediately from section 5 on.
At last, we derive the conditions for *P* in order that $\mathcal{A}$ can be finite and still characterizes *P*.

3. Preliminaries on program schemes

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

A *program scheme* P is a ordered pair $P \equiv \langle D,S \rangle$, where $D \equiv \{P_1 \Leftarrow S_1, \dots, P_n \Leftarrow S_n\}$ is a *declaration scheme* and S and the *bodies* $S_1, \dots, S_n$ are *statement schemes*, which are inductively constructed from

- *elementary statement symbols* (A, with indices), also called *relation symbols*
- *procedure symbols* (P, with indices)
- *constant symbols (identity symbol* E, *emptyness symbol* Ω*)*

by means of binary compositions with pairs of parentheses and

- the *alternation symbol* $\cup$
- the *sequention symbol;*

Where no confusion results we omit the parentheses and in particular we allow the *empty string* Λ as the trivial statement scheme.

A meaning (*interpretation*) $c(P)$ for a program scheme $P \equiv \langle D,S \rangle$ is the interpretation $c(S)$ of the statement scheme S under an interpretation c with respect to the declaration scheme D.

An *interpretation* c consists of the choice of a set, the *domain* Dom(c), and the choice of a *binary relation* c(A) on Dom(c) for each relation symbol A.

A statement scheme S *is interpreted* under an interpretation c with respect to a declaration scheme D as the binary relation c(S) on Dom(c), defined as:

> $(x,y) \in c(S)$ if there exists a *computation sequence* $x_0(S_0)x_1(S_1)\dots x_n(S_n)x_{n+1}$, with $x = x_0$, $S \equiv S_0$ and $S_{n+1} \equiv \Lambda$, $x_{n+1} = y$ and for $i = 0,1,\dots,n$ the *intermediate results* x_i in Dom(c) and the *components* S_i, statement schemes, satisfy the following seven conditions (six of which are obtanined by either reading or skipping the symbols between [and]):

- if $S_i \equiv (S';S'');S'''$ then $S_{i+1} \equiv S';\ (S'';S''')$ and $x_{i+1} = x_i$
- if $S_i \equiv (S' \cup S'')\ [;S''']$ then $S_{i+1} \equiv$ either $S'[;S''']$ or $S''\ [;S''']$and $x_{i+1}=x$
- if $S_i \equiv P\ [;S''']$ then $S_{i+1} \equiv S\ [;S''']$ and $x_{i+1} = x_i$ where $P \Leftarrow S$ is in D
- if $S_i \equiv L\ [;S''']$ then $S_{i+1} \equiv [S''']$ and $(x_i,x_{i+1}) \in c(L)$ for relation and constant symbols L.

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

A statement scheme S *occurs executably* in a computation sequence if there is a component $S_i \equiv S\ ;S'$.

Define an operation " ; " for relations to be the *concatenation*, $(x,y) \in R_1;\ R_2 \ \underset{\text{def}}{\leftrightarrow}\ \exists z\colon (x,z) \in R_1 \wedge (z,y) \in R_2$, and an operation "∪" to be the *union* of relations. Then ; and ∪ are associative and for schemes S_1 and S_2 we have ,for all interpretations c, $c(S_1;S_2) = c(S_1);\ c(S_2)$ and $c(S_1 \cup S_2) = c(S_1) \cup c(S_2)$. Furthermore, $c(\Omega) = c(E)$.

3.2. 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 ⊆. The connective = is used for the equality on the domain of the interpretation, both for elements and for relations. Thus $S_1 = S_2$ is short for $S_1 \subseteq S_2,\ S_2 \subseteq S_1$.

The connectives ≡ and $\underline{\subseteq}$ are used to express the (stronger) syntactical relations with respect to formal languages and pure formal objects.

3.3. The formalism thus defined, of program schemes and assertions about interpretations of them has great powers of expression.

It handles *recursiveness*.

It handles *nondeterminism* by means of (the interpretation of) the alternation symbol ∪ and the interpretation of the elementary statement symbols as relations rather than functions.

Predicate symbols (B, $\neg B$, P_{in}, P_{ex}, p...) are considered as a special class of relation symbols by the fact that their interpretation is a subrelation of the identity relation. So let "B" be an predicate, then
$B = \{(x,x) \mid$ "B" holds for $x\}$ is identified with "B", and
$\neg B = \{(x,x) \mid$ "B" does not hold for $x\}$ is identified with not-"B".
Thus we can denote the conditional$(B \rightarrow A_1, A_2)$ or if B then A_1 else A_2 by $B;\ A_1 \cup \neg B;\ A_2$.

Correctness assertions $\forall x,y\ [P_{in}(x) \wedge x\ P\ y \rightarrow P_{ex}(y)]$ give a property which holds for the output of a program provided that the input satisfies some initial property.Hoare writes them as $P_{in}\{P\}\ P_{ex}$ but in our formalism they read $P_{in};P \subseteq P;\ P_{ex}$. *Inductive assertions* are such assertions about the parts from which the program - inductively - is constructed. We restrict inductive assertions to inclusions $p;A \subseteq A;q$ for *elementary* statement symbols A. A set of inductive assertions is called an inductive assertion pattern.

The while *statement* while B do A is due to ist meaning in fact a call of the recursive procedure $W \Leftarrow B;A;W \cup \neg B$. The proof rule of Hoare reads: $\quad p \wedge B\{A\}p \quad$ implies $\quad p\{W\}p \wedge \neg B$
or: $\forall x,y[p(x) \wedge B(x) \wedge xAy \rightarrow p(y)]$ implies $\forall x,y\ [p(x) \wedge xWy \rightarrow p(y) \wedge \neg B(y)]$
In our formalism it becomes $p;B;A \subseteq A;p \models p;W \subseteq W;p;\neg B$.

4. Program schemes and grammars

4.1. Given a program scheme $P \equiv \langle D,S_0\rangle$ with $D \equiv \{P_1 \Leftarrow S_1,\ldots,P_n \Leftarrow S_n\}$, then we define a context-free grammar G associated with P as follows.
Let P_0 be a new procedure symbol and set $D_0 \equiv \{P_0 \Leftarrow S_0\} \cup D$ and $P_0 \equiv \langle D_0,P_0\rangle$. Now $G \equiv \langle V_N,V_T,R,P_0\rangle$, where
V_N is the set of procedure symbols in P_0 united with a set of symbols, namely for each statement scheme occurring in P_0 of the form $(S_1 \cup S_2)$ one symbol which we write as $(S_1 \cup S_2)$,
V_T is the set of elementary statement symbols, also called relation symbols,
R is a set of derivation rules;
for each declaration $P_i \Leftarrow S_i$ in D one rule which we write as $P_i \rightarrow S_i$, and for each statement scheme $(S_1 \cup S_2)$ in P_0 two rules, which we write as $(S_1 \cup S_2) \rightarrow S_1$ and $(S_1 \cup S_2) \rightarrow S_2$.

In sentential forms we explicitly denote the concatenation operator as; .
Due to the notation of the symbols of G, syntactical objects can be considered as schemes and vice versa. Thus for $\tau \in L(G)$ and interpretation c, $c(\tau)$ makes sense and for a statement scheme S, $S \in L(G)$ makes sense.

Convention. We use $\sigma,\sigma_1,\ldots$ as variables over $(V_N \cup V_T)^*$, N over V_N, τ over V_T^* and, of course, A over V_T.
For interpretations c and a set $L \subseteq V_T^*$ we let
$c(L) =_{def} \bigcup_{\tau \in L} c(\tau)$, in particular
$c(L(G)) =_{def} \bigcup_{\tau \in L(G)} c(\tau)$.

In the next two lemmas we do not prove, at least not explicitly, the syntactical equality of the evaluation sequences of P and the sentences of the grammar G associated with P, but a semantical consequence of it, namely $\models P = L(G)$.

4.2. Lemma

For any interpretation c and x and y in Dom(c) and any $\tau \in L(G)$ such that $(x,y) \subset c(\tau)$
there exists a computation sequence $x(P_0)$--- y (w.r.t. D_0 and c) .

proof

By induction on the length of the left most derivations in G, denoted by $\Longrightarrow^*$, we prove:

for all $\sigma \Longrightarrow^* \tau_1\sigma_1$ where σ and σ_1 do not begin with a relation symbol and $(x,y) \in c(\tau_1)$
there is an initial comp. seq. $x(\sigma)$ --- $y(\sigma_1)$ w.r.t. D_0 and c.

The lemma then follows as a special case with $\sigma \equiv P_0$ and $\sigma_1 \equiv \Lambda$.

length = 0: the trivial case, no derivation, $\sigma \equiv \tau_1\sigma_1$ and $\tau_1 \equiv \Lambda$.
Hence $c(\tau_1) = c(E)$ and x is identical to y. Then x alone is the initial comp. seq., it is a trivial one with length zero.

length > 0: let $\sigma \Longrightarrow^* \tau_1 \; N \; \sigma_1 \Longrightarrow \tau_1\tau_2\sigma_2\sigma_1$ where $\sigma_2\sigma_1$ does not begin with a relation symbol ~~and~~ $(x,y) \in c(\tau_1;\tau_2)$.
Then there is a z such that $(x,z) \in c(\tau_1)$ and $(z,y) \in c(\tau_2)$.
For $\sigma \Longrightarrow^* \tau_1 \; N \; \sigma_1$ with $(x,z) \in c(\tau_1)$ there is by induction hypothesis an initial comp. seq. $x(\sigma)$ --- $z(N\sigma_1)$.
We let an extension of this initial comp. seq. correspond to the applied rule $N \rightarrow \tau_2\sigma_2$ in the derivation. Namely, in case N is a symbol $(S_1 \cup S_2)$ the rewriting of N as the alternative S_1 or S_2 whichever is $\tau_2\sigma_2$, and in case N is a procedure symbol the rewriting of N as the body (which is $\tau_2\sigma_2$).
Let $\tau_2 \equiv A_1; \; \ldots; \; A_p$ then there are $z \equiv z_0, z_1, \ldots, z_p \equiv y$
with $(z_{i-1}, z_i) \in c(A_i)$. We extend the comp. seq. to
$x(\sigma)$ --- $z(N\sigma_1)z_0(A_1;\ldots;A_p;\sigma_2\sigma_1)z_1(A_2;\ldots;A_p;\sigma_2\sigma_1)$---- $z_p(\sigma_2\sigma_1)$,
so that the induction step has been proved.

4.3. Lemma

For every comp. seq. $x(P_0)$ --- y w.r.t. D_0 and interpretation c there exists a $\tau \in L(G)$ such that $(x,y) \in c(\tau)$.

proof

We prove by induction on the number of components not beginning with a relation symbol that for every comp. seq. $x_0(\sigma)$ --- x_k, where σ does not begin with a relation symbol,

there exists a left most derivation $\sigma \overset{*}{\Longrightarrow} \tau$ for G with $(x,y) \in c(\tau)$.

The lemma then follows as a special case with $\sigma \equiv P_0$.

number = 0: trivial case, the "empty" sequence x_0 where $\sigma \equiv \Lambda$.

The required derivation is the trivial empty one, where $\sigma \equiv \Lambda \equiv \tau$, for $c(\Lambda) = c(E)$ so $(x_0,x_0) \in c(\tau)$.

number > 0: let σ_1 be the first component after σ not beginning with a relation symbol. Then the sequence is $x_0(\sigma)$ --- $x_l(\sigma_1)$ --- x_k. (if number = 1 then $\sigma_1 \equiv \Lambda$ and $l = k$).

By induction hypothesis there exists a derivation $\sigma_1 \overset{*}{\Longrightarrow} \tau_1$ with $(x_l,x_k) \in c(\tau_1)$.

The initial segment of the sequence has the form

$x_0(N\sigma')x_0(A_1;\ldots;A_l;\sigma_1)x_1(A_2;\ldots;A_l;\sigma_1)$ --- $x_l(\sigma_1)$ with $(x_0,x_l) \in c(A_1;,..;A_l)$. To the rewriting of N as $A_1;A_2\ldots$. (that is to say, either the body of N if N is a procedure symbol or an alternative if N is a scheme $(S_1 \cup S_2)$)there correponds a derivation rule in G (namely $P \to S$ with $S \equiv A_1;A_2\ldots$ or $(S_1 \cup S_2) \to S_1$ or $(S_1 \cup S_2) \to S_2$, depending on whether $S_1 \equiv$ or $S_2 \equiv A_1;A_2\ldots$). So

$\sigma \equiv N\sigma' \overset{1}{\Longrightarrow} A_1;\ldots;A_l;\sigma_1 \overset{*}{\Longrightarrow} A_1;\ldots;A_l;\tau_1$ is a derivation in G with $(x_0,x_l) \in c(A_1;\ldots;A_l)$ and $(x_l,x_k) \in c(\tau_1)$, so $(x_0,x_k) \in c(A_1;\ldots;A_l;\tau_1)$.

Hence the induction step has been proved.

4.4. Theorem $\models P = L(G)$

proof

Let c be any interpretation and x and y in Dom(c). Then by definition $(x,y) \in c(P)$ if and only if there exists a comp. seq. $x(P_0)$ --- y w.r.t. D_0 and c. Now apply the lemmas.

5. Inductive assertion patterns and tied complete derivation trees

In order to get an intuitive idea somewhat easier, the reader may follow the historical development as outlined in section 2.
Therefore he should then first read the text up to (6.6), skipping the additions between the square brackets [and], concerning "tying the trees". After that he may read the complete text form here onwards.

5.1. As described in section 2 we want to give the definition of an inductive assertion pattern on the basis of a [tied] complete derivation tree.
For the sake of clarity we need some terminology.

On the first reading of def. (5.2.) we advise the reader to realize that the "derivation trees" without the additions between the brackets { and }, concerning the completeness, precisely cover the usual concept from formal language theory. The noun "occurrence" reflects the fact that we have in mind a diagram as intuitive image of derivation trees.
After tying a tree it is rather strange to speak of subtrees and therefore we denote that concept with the noun "component". As denotation of the *symbol* whose occurrence is the root of a tree (component), the noun "original" is used. We use the noun "origin" instead of *"root"*, in order to render the close connection with the original.

Definitions

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

- a [not necessarily] new occurrence of the symbol S, where the occurrence of S is called the *origin* of $\mathcal{B}$,
- with in case S is a terminal symbol nothing but in case S is a nonterminal symbol:

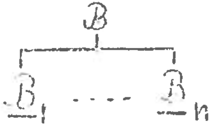
  a by this occurrence of S uniquely determined *sequence* {resp. uniquely determined collection of uniquely determined sequences} of [tied] {complete} derivation trees, that is to say

  > 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).

5.2.* Examples

We denote the sequences (of direct components $\mathcal{B}_i$, see 5.3.) of $\mathcal{B}$ schematically by

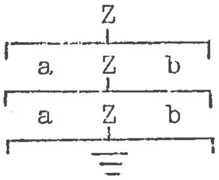


,where $\mathcal{B}_i$ is the origin of $\mathcal{B}_i$ or a reference "□↗" to it.

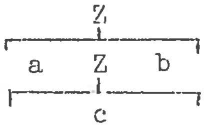
Let a grammar be given by Z → aZb, Z → c

a) then some not-tied derivation trees for Z are:

$\mathcal{B}'$:

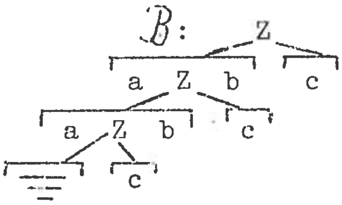


$\mathcal{B}''$:



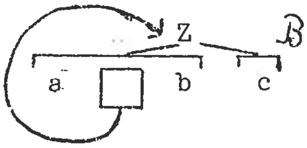
b) and the not-tied complete der. tree for Z is:

$\mathcal{B}$:

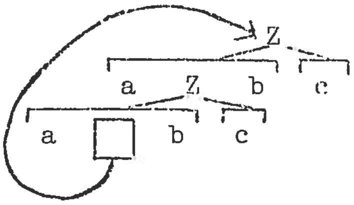


[c) and some tied complete der. trees for Z are:

$\mathcal{B}_1$



$\mathcal{B}_2$



]

Definitions

5.3. For a [tied] {complete} derivation tree $\mathcal{B}$:

A *(left most/right most) direct component*

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

A *(left most/right most) component*

is the tree $\mathcal{B}$ itself, and

every (left most/right most) component of any (left most/right most) direct component of $\mathcal{B}$.

(Directly) successive are two *direct components* 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 *components* X and Y

if there are two direct components U and V of a component of $\mathcal{B}$, such that

U and V are (directly) successive and

X is (right most) component of U and Y is (left most) component of V.

5.3.* Examples

We will denote the components by their origins. Refering to example 5.2.* and restricting to the components which are occurrences of terminal symbols, we can state for $\mathcal{B}$ that

- the uppermost a is left most component and the remaining a's are directly successive to the a occurring one level higher.
  Analogously for the b's.
- the uppermost c is both left most and right most comp., and the remaining c's are directly successive to the a occurring one level higher and are directly succeeded by the b occurring one level higher

[and for $\mathcal{B}_1$ that

- the a is both left most comp. and direct successor of itself. Analogously for b.
- the c is both left most and right most comp., and is direct successor of the a and is directly succeeded by the b.]

Definitions

5.4. A *(direct) production*

is a sequence of directly successive (direct) components $\mathcal{B}_1;\ldots;\mathcal{B}_n$ of $\mathcal{B}$, such that $\mathcal{B}_1$ and $\mathcal{B}_n$ are respectively a left most and a right most component of $\mathcal{B}$.

A *terminal component*

is a component with a terminal symbol as original.

A *terminal production*

is a production consisting of terminal components.

A *sentential form* (resp. *sentence*)

is the sequence of originals of a (terminal) production.

The *language* $L(\mathcal{B})$ of $\mathcal{B}$

is the set of sentences of $\mathcal{B}$.

We denote a production with sentential form σ by $\hat{\sigma}$ and a component with original S by $\hat{S}$.

5.4.* Examples

Refering to 5.2.* , $L(\mathcal{B}) \equiv \{a^n cb^n \mid n \geq 0\}$ [and $L(\mathcal{B}_1) \equiv \{a^m cb^n \mid m,n \geq 0\}$ and $L(\mathcal{B}_2) \equiv \{a^m cb^n \mid n,m \geq 0$ and $n = m \pmod 2\}$].

From the definitions follows immediately the

5.5. Theorem

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

proof is left to the reader.

5.6. The definition

of the Inductive Assertion Pattern $\mathcal{A}$ based upon a [tied] complete derivation tree $\mathcal{B}$ and with respect to correctness predicate symbols P_{in} and P_{ex} reads:

Let $\{P_i\}_{i \in I}$ be a collection of new predicate symbols which are bijectively associated with the terminal components of the tree $\mathcal{B}$.

Then $\mathcal{A}$ consists of the following assertions

- for directly successive terminal components $\hat{A}$ and $\hat{A}'$ (with associated predicatedsymbols p and p') the inclusion $p;\ A \subseteq A;\ p'$
- for left most and right most terminal components $\hat{A}$ and $\hat{A}'$ (with predicate symbols p and p') the inclusions $P_{in} \subseteq p$ resp. $p';A' \subseteq A';\ P_{ex}$

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 $\mathcal{C}(..)$ is an abbreviation of $p_{in};..\subseteq..;\ p_{ex}$.

[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

6.1. Main lemma

(i) $\mathcal{A} \models \mathcal{C}(L(\mathcal{B}))$

(ii) for any scheme T

$\mathcal{A} \models \mathcal{C}(T)$ implies $\models T \subseteq L(\mathcal{B})$

(iii) $\mathcal{C}(L(\mathcal{B})) \models_{*} \mathcal{A}$

(*: ~~provided~~ that the pred. symbols $P_i (i \in I)$ in $\mathcal{A}$ are conveniently interpreted).

proof

(i) Let c be any interpretation and let $\tau \equiv A_1;\ldots;A_n \in L(\mathcal{B})$. Then there are directly successive terminal components $\hat{A}_1,\ldots,\hat{A}_n$ in $\mathcal{B}$, of which $\hat{A}_1$ and $\hat{A}_n$ are left most and right most ones. Consequently the inclusions $p_{in} \subseteq p_1$ and $p_i;A_i \subseteq A_i;p_{i+1}$ for $i = 1,\ldots,n-1$ and $p_n;A_n \subseteq A_n;p_{ex}$ belong to $\mathcal{A}$. From the validity of $\mathcal{A}$ under interpretation c there follows consecutively for $i = o,1,\ldots,n-1$ the validity under c of

$p_{in};A_1;\ldots;A_i \subseteq A_1;\ldots;A_i;p_{i+1}$ so

$p_{in};A_1;\ldots;A_i;A_{i+1} \subseteq A_1;\ldots;A_i;p_{i+1};A_{i+1}$ so according to $p_{i+1};A_{i+1} \subseteq A_{i+1};p_{i+2}$

$p_{in};A_1;\ldots;A_i;A_{i+1} \subseteq A_1;\ldots;A_i;A_{i+1};p_{i+2}$

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

$\mathcal{A} \models_{c} p_{in};\tau \subseteq \tau;p_{ex}$ for any $\tau \in L(\mathcal{B})$, so also

$\mathcal{A} \models_{c} \bigcup(p_{in};\tau) \subseteq \bigcup(\cancel{p_{in}};\tau;p_{ex})$ the union taken over all $\tau \in L(\mathcal{B})$,

$\mathcal{A} \models_{c} p_{in};(\bigcup\tau) \subseteq \cancel{p_{in}};(\bigcup\tau);p_{ex}$ Q.E.D.

(ii) Let c be any interpretation, we show $\models_c \; T \subseteq L(\mathcal{B})$. Thus let x_0 and y_0 be arbitrary elements in $\mathcal{D}om(c)$ 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 form 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 occurr 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_0 and a computation by the successive $A_1,\dots,A_{i-1}$ (which occurr 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. Here in 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

$(x,x) \in c'(p) \leftrightarrow_{def}$ there are directly successive terminal components $\hat{A}_1,\dots,\hat{A}_i$ in $\mathcal{B}$ such that $\hat{A}_1$ is left most component and $(x_0,x) \in c(A_1;\dots;A_{i-1})$ and p is associated with $\hat{A}_i$.

$(x,x) \in c'(p_{in}) \leftrightarrow_{def} x = x_0$

$(x,x) \in c'(p_{ex}) \leftrightarrow_{def} (x_0,x) \in c(A_1;\dots;A_n)$ for some terminal production $\hat{A}_1;\dots;\hat{A}_n$ of $\mathcal{B}$.

It is now easy to verify the validity of $\mathcal{A}$ under c':

- if p; $A \subseteq A$; p' belongs to $\mathcal{A}$, then p and p' are associated with directly successive terminal components $\hat{A}$, $\hat{A}'$. So from $(x,y) \in c'(p;A)$ it follows that $(x,y) \in c'(A)$ <u>and</u> $(x_0,x) \in c(A_1;\dots;A_{i-1})$ for some sequence of directly successive terminal components $\hat{A}_1,\dots,\hat{A}_{i-1},\hat{A}_i \equiv \hat{A}$. So we have $(x,y) \in c'(A)$ <u>and</u> $(x_0,y) \in c(A_1;\dots;A_{i-1});c(A_i) = c(A_1;\dots;A_i)$, 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 had assumed $(x_0,y_0) \in c(T)$ so that $(x_0,y_0) \in c'(p_{in};T)$ holds, too. So with the inclusion just derived it follows that $(x_0,y_0) \in c'(T;p_{ex})$ and in particular $(y_0,y_0) \in c(p_{ex})$. By definition this means $(x_0,y_0) \in c(\tau)$ for some sentence $\tau \in L(\mathcal{B})$, hence $(x_0,y_o) \in c(L(\mathcal{B}))$. 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

$(x,x) \in c(p) \leftrightarrow_{def}$ there is some x_0 in Dom(c) such that $(x_0,x) \in c(p_{in};A;\ldots;A_{i-1})$ for an initial segment $A_1;\ldots;A_i$ of some sentence of $\mathcal{B}$ where p is associated with $\hat{A}_i$.

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 $p_{in} \subseteq p$ and $p;A \subseteq A;p'$ 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.

In order to give some conditions in the next lemma and theorem by a syntactical rather than a semantical inclusion, we need the following

6.2. Lemma

For languages L_1 and $L_2 \subseteq V_T^*$: $\models L_1 \subseteq L_2$ if and only if $L_1 \subseteq L_2$.

proof

From right to left trivial. From left to right:let $\tau \in L_1$, say $\tau \equiv A_1;\ldots;A_n$, then we have to show $\tau \in L_2$. Consider the lefthand side with a special interpretation c ("Herbrand") as follows: $Dom(c) \equiv V_T^*$ i.e. the strings over the symbols A_i and $c(A_i)$ given by $(x,y) \in c(A_i) \leftrightarrow_{def} x;A_i \equiv y$ (note that $x;A_i$ is the concatenation of the string $x \in V_T^*$ with the symbol $A_i \in V_T$). The lefthand side states that for arbitrary x and y in V_T^*

$(x,y) \in c(\tau)$ implies $(x,y) \in c(\tau')$ for some $\tau' \in L_2$. But $(x,y) \in c(\tau')$ if and only if $x;\tau' \equiv y$, so with $x;\tau \equiv y$ we get $\tau \equiv \tau' \in L_2$.

6.3. Lemma

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

(ii) for any scheme T

$\mathcal{A} \models \mathcal{C}(T)$ implies $\models T \subseteq P$ iff $L(\mathcal{B}) \subseteq L(G)$

(iii) $\mathcal{C}(P) \models_{*} \mathcal{A}$ iff $L(\mathcal{B}) \subseteq L(G)$

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

proof

We tacitly apply theorem 4.4. $\models P = L(G)$ and lemma 6.2. where $L_1 \equiv L(\mathcal{B})$ and $L_2 \equiv L(G)$. M1 is an abbreviation of "Main lemma".

(i) $\Rightarrow$: substitute P and $L(G)$ for T in M1(ii).

$\Leftarrow$: note that for all c $\forall$p,q $\forall$ relations $R_1 \subseteq R_2$ on Dom(c):
$c(p);R_2 \subseteq R_2;c(q) \rightarrow c(p);R_1 \subseteq R_1;c(q)$ (prove it elementswise).
Since $P = L(G) \subseteq L(\mathcal{B})$ we may substitute P for $L(\mathcal{B})$, in M1(i).

(ii) $\Rightarrow$: substitute in the implication $L(\mathcal{B})$ for T. The premiss of the resulting implication holds, see M1(i). Hence $L(\mathcal{B}) \subseteq P = L(G)$.

$\Leftarrow$: apply transitivety of the inclusion, and $L(\mathcal{B}) \subseteq L(G)$, to M1(ii).

(iii) $\Rightarrow$: consider the lefthand side under the interpretation c of lemma 6.2., extended with $(x,x) \in c(p_{in}) \leftrightarrow_{def} x \equiv \Lambda$ (empty string) and $(x,x) \in c(p_{ex}) \leftrightarrow_{def} x \equiv A_1;\ldots;A_n$ for some sentence of $L(G)$ and for the $p_i (i \in I)$ in $\mathcal{A}$ anything, for instance the convenient interpretation as indicated in M1(iii). By the choice of c, $\mathcal{C}(L(G))$ so $\mathcal{C}(P)$ is valid under c. Hence according to the lefthand side, $\mathcal{A}$ too. The validity of $\mathcal{C}(L(\mathcal{B}))$ under c follows from M1(i) and this implies $L(\mathcal{B}) \subseteq L(G)$ by the choice of c.

$\Leftarrow$: if $L(\mathcal{B}) \subseteq L(G)$ then the validity of $\mathcal{C}(P)$ under c implies the validity of $\mathcal{C}(L(\mathcal{B}))$ under c, c.fr (i)$\Leftarrow$. Now apply M1(iii).

6.4. Characterization theorem

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

(i) for any scheme T

$\mathcal{A} \models \mathcal{C}(T)$ 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

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

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

proof

Suppose $L(\mathcal{B}) \equiv L(G)$, i.e. both $L(\mathcal{B}) \subseteq L(G)$ and $L(\mathcal{B}) \supseteq L(G)$. Then

(i) $\Longrightarrow$:follows from lemma (6.3.ii). $\Longleftarrow$: follows from lemma (6.3.i) together with the remark in the proof of (6.3.i$\Longleftarrow$).

(ii) It is proved (see[I]) that P is the minimal fixed point, i.e. $\models P \subseteq T$ for arbitrary fixed point T of the scheme P. This together with (i) completes the proof.

(iii) Lemma (6.3.i) states the justification of the proof rule. Suppose that some correctness assertion $\mathcal{C}(P)$ is valid under some interpretation c, then (6.3.iii) states that $\models_c *\mathcal{A}$ holds, hence the premiss of the proof rule $\mathcal{A} \models_c \mathcal{C}(P)$ is valid. Thus $\mathcal{C}(P)$ could have been obtained by application of the proof rule.

Suppose that either (i) or (ii) or (iii) holds. We show $L(\mathcal{B}) \equiv L(G)$. Each of (i)/(ii)/(iii) implies the lefthand sides of both lemma (6.3.i) and lemma (6.3.ii/ii/iii). Hence both $L(\mathcal{B}) \supseteq L(G)$ and $L(\mathcal{B}) \subseteq L(G)$.

6.5. Conclusion. In order that the Inductive Assertion Pattern $\mathcal{A}$ - based upon some [tied] complete derivation 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 theorem 5.5. 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.

[Please continue reading at 6.6] The reader who has skipped the additions may now read the complete text from 5.1. onwards.

6.6. Now we give the conditions for G such that $\mathcal{B}$ can be chosen finite and $\mathcal{A}$ still characterizes P

<u>Finiteness Theorem</u>

$\mathcal{B}$, 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).

<u>proof</u>

$\mathcal{A}$ is finite precisely when $\mathcal{B}$ is finite (in the number of occurrences of terminal symbols). Let $\mathcal{B}$ be finite and $L(\mathcal{B}) \equiv L(G)$. We define a non deterministic finite automaton which accepts $L(\mathcal{B})$, by considering $\mathcal{B}$ as such. The input symbols are the originals of terminal components, i.e. V_T. The states are the origins of the terminal components of $\mathcal{B}$ and also the origin of $\mathcal{B}$ itself. The initial state is the origin of $\mathcal{B}$ and the final states are the origins of rightmost terminal components of $\mathcal{B}$. From a state, i.e. the origin of a terminal component, transitions are possible to each origin of a directly successive terminal component by the input ~~symbol~~ which is the original of that component.
Clearly $L(\mathcal{B})$ is the language accepted by the automaton, so $L(\mathcal{B}) \equiv L(G)$ is regular. *)

Conversely, let $L(G)$ be regular. Consider a regular grammar G' with $L(G') \equiv L(G)$. We give a construction for a finite tied complete derivation tree $\mathcal{B}$ for the sentence symbol P_o' of G' such that $L(\mathcal{B}) \equiv L(G')$.
Construct $\mathcal{B}$ inductively according to defintion 5.2. with respect to G', beginning with a new occurrence of P_0' as the origin and further choosing already existing occurrences of nonterminal symbols for the origins of direct components, whenever it is possible and otherwise a new occurrence. The tree $\mathcal{B}$ thus constructed is finite.
By induction on the lenght of the derivations it is easy to prove that for every derivation $P_o' \overset{*}{\Rightarrow} \sigma$ in G' there is a production $\hat{\sigma}$ in $\mathcal{B}$ which has the sentencial form σ. Hence $L(G') \subseteq L(\mathcal{B})$.
Also $L(G') \supseteq L(\mathcal{B})$ holds. For due to the regularity of G' the components $\hat{N}$, N any nonterminal symbol, have no direct successive components. So the productions of $\mathcal{B}$ are of the form $\hat{A}_1;\ldots;\hat{A}_n;\hat{N}$ or $\hat{A}_1;\ldots;\hat{A}_n$. From the regularity and the definition of direct successiveness it follows that in those productions $\hat{A}_n;\hat{A}$ resp. $\hat{A}_n$ is the sequence of direct components of some $\hat{N}'$ such that $\hat{A}_1;\ldots;\hat{A}_{n-1};\hat{N}'$ is a production. And hence $N' \to A_n;N$ resp. $N' \to A_n$ is in G'. Repeatedly applying this step renders a derivation in G' for every (terminal) production, and hence for every sentence of $\mathcal{B}$. Q.E.D.

*) <u>footnote</u> see last page

6.7. We have given a very general definition of tying trees. In 6.6. 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 [I, section 4.1.] which appeared to fail.
There some patterns were proposed as characterizing a recursive procedure P, but whereas $\mathcal{A} \models \mathcal{C}(P)$ was true, even the implication "$\mathcal{A} \models \mathcal{C}(T)$ implies $\models T \subseteq P$" 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.
In general L($\mathcal{B}$') $\supseteq$ L(G') does hold for a tree $\mathcal{B}$' set up for a grammar G' (proof: left to the reader). So it follows from the Mainlemma and lemma 6.2. that we can find counter examples T for the above implication as sentences $\tau \in$ L($\mathcal{B}$) — L(G-of-P), or even via theorem 4.4. as program schemes T of which the associated grammar G-of-T produces a languages with L(G-of-P) $\not\supseteq$ L(G-of T) ∧ L(G-of-T) $\subseteq$ ~~L($\mathcal{B}$).~~
Moreover, each counter example should satisfy this requirement.

7. Example/Application: the while statement characterization

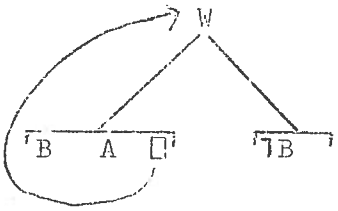
The while statement $W \equiv$ while B do A can be considered as recursively defined by $W \Leftarrow B;A;W \cup \neg B$.

The grammar associated to W is given by

$W \to (B;A;W \cup \neg B) \quad (B;A;W \cup \neg B) \to B;A;W \quad (B;A;W \cup \neg B) \to B$

An equivalent grammar in regular form is

$W \to B;A;W \quad W \to \neg B$



The tree $\mathcal{B}$ constructed according to 6.6. looks like

The pattern $\mathcal{A}$ based upon $\mathcal{B}$, is according to definition 5.2.:

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

By contracting the 2rd + 3rd and the 2nd + 4th inclusion we can eliminate p_A. Thus we obtain the equivalent*) pattern

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

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

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

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

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

According to theorem 6.6. and 6.4. we have that

(⊛) (i) for any scheme T

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

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

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

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

and $p_{in};W \subseteq W;p_{ex} \models_* \mathcal{A}'''$

): 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).

The while statement characterization given above may seem to differ from the characterization mentioned in the introduction 1.1.
However, the latter can be formulated as

(&&) (i) for any scheme T

$\mathcal{A}^{\circ} \models p;T \subseteq T;p;\neg B$ if and only if $\models T \subseteq W$

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

$\mathcal{A}^{\circ} \models p;T \subseteq T;p;\neg B$ if and only if $\models T = W$

(iii) both $\mathcal{A}^{\circ} \models p;W \subseteq W;p;\neg B$

and $r;W \subseteq W;q \models r \subseteq p, \mathcal{A}^{\circ}, p;\neg B \subseteq q$

where

$\mathcal{A}^{\circ} = \{p;B;A \subseteq A;p\}$.

Now, the only difference is that a suitable instance of the additional rule (see 1.1) "$R \Rightarrow P$, $P\{T\}S$, $S \Rightarrow Q$ implies $R\{T\}Q$" is already implemented in version (&), whereas in version (&&) it is only implemented in the second part of (iii). The equivalence of (&) and (&&) can easily be demonstrated by suitable substitutions for the predicate symbols.

footnote$^{*)}$ ad theorem 6.6

It is the very first clause of def. 5.4. which induces a control mechanism for the production of sentences that can be realized by some finite automaton, namely the one just constructed. This in contrast with a variant, like:

a production π of $\mathcal{B}$

is - the tree $\mathcal{B}$ itself and

- every concatenation $\pi_1;\ldots;\pi_n$ of productions π_i of $\mathcal{B}_i$ where $\mathcal{B}_1,\ldots,\mathcal{B}_n$ is a sequence of direct components of $\mathcal{B}$, as mentioned in 5.2.

The control mechanism induced by this clause is realized by a push-down automaton. With such a definition $\mathcal{B}$ produces precisly L(G) where G is the underlying grammar, even in the case of tying.

We did not choose this latter variant, because the essential proof, Mainlemma (6.1.ii and iii), is based upon the former control mechanism.

Refenrences

[I] de Bakker, Meertens:
Simple recursive program schemes and inductive assertions
MR142/72 DECEMBER, Math. Centre, Amsterdam

The same program schemes as used above occurr in

[II] de Bakker:
Recusive procedures
Math. Centre Tracts 24, Math. Centre, Amsterdam 1971

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

[IV] Hoare:
An axiomatic basis for computer programming, C.ACM 12 pp.576-583(1969)

Recently: de Bakker, J.W & Meertens, L.G.T.L, :
On the completeness of the ind. ass. method
to appear.