Correctness Proofs of Syntax-Directed Processing Descriptions by Attributes

C. Pair,* M. Amirchahi,† AND D. Neel†

* Centre de Recherche en Informatique, B.P. 3308, 54014 Nancy, France; † Iria-Laboria, B.P. 105, 78150 Le Chesnay, France

Received December 10, 1976; revised October 2, 1978

A method is presented for correctness proofs of text processing descriptions, using Knuth's semantic attributes. The method is applied to examples dealing with compiler optimizations and in particular to the equivalence demonstration of descriptions by attributes.

INTRODUCTION

Numerous publications [6, 9, 12] have shown that the use of semantic attributes could be an excellent tool for describing certain text processings, in particular for the translation of programming languages. In fact, if the attributes method sometimes adds a burdensome factor to compilation processes, this drawback is compensated by an increase in clarity and more rigor in formalism, specially significant where the task consists of defining a compiler that others would have to implement and still another group would eventually use. Nevertheless, for the method to be satisfactory, some proof techniques have to be associated with the attributed description. This, to our knowledge, has not yet been done, although correctness of compilers has been dealt with by numerous authors, e.g., [1, 2, 4, 7, 8, 11], using tools such assertions, denotational semantics, LCF, operational semantics . . . . Generally the correctness is proved by a bottom-up induction (or structural induction) on the syntax tree of the program. But attribute grammars leave the order of evaluation of semantic functions largely unspecified and therefore a correctness proof method need not be limited to bottom-up induction.

In the first section of this paper, the reader is initiated to the attributes method using an example taken from [6] where nonexecuted statements in a program are determined.

Section II introduces a method for the proof of definitions by attributes, applied also to the preceding example. The proof method, based on induction, is made precise in Section III.

In Section IV, the same principle is applied to another example illustrating the equivalence of two semantic descriptions, one optimized (the optimization concerning elimination of common redundant subexpression [7]) and the other nonoptimized.

Before the conclusion, a special and simple case of the proposed method is displayed, viz., proof by synthesized induction.
I. SEMANTIC ATTRIBUTES

1.1. Attribute Notion

The notion of attributes was developed by D. E. Knuth in 1968 [2] to describe formally the semantic aspects of languages defined by a context-free grammar.

An attribute confers some "semantic" information to a node of the syntax tree of a sentence. Certain attributes are defined as a function of attributes of the "father" node in the syntax tree: They are said to be inherited; others are evaluated from attributes of descendant nodes: Hence they are said to be synthesized.

1.2. Example of an Attributed Description

The extremely simplified language in which our examples are formulated is defined by the following grammar rules:

\[
\begin{align*}
\langle P \rangle &::= \langle L \rangle \\
\langle L \rangle &::= \langle S \rangle \\
\langle L \rangle &::= \langle L \rangle \;; \langle S \rangle \\
\langle S \rangle &::= \text{Label} \::\! \langle NLS \rangle \\
\langle S \rangle &::= \langle NLS \rangle \\
\langle NLS \rangle &::= \langle V \rangle = \langle E \rangle \\
\langle NLS \rangle &::= \langle DI \rangle \;; \langle L \rangle \;; \text{END} \\
\langle DI \rangle &::= \text{DO} \langle V \rangle - \langle E \rangle \text{ TO } \langle E \rangle \text{ BY } \langle E \rangle \\
\langle E \rangle &::= \langle O \rangle \\
\langle E \rangle &::= \langle E \rangle + \langle O \rangle \\
\langle O \rangle &::= \langle V \rangle \\
\langle O \rangle &::= \text{Constant} \\
\langle V \rangle &::= \text{Identifier}
\end{align*}
\]

\(\langle P \rangle\): Program, \(\langle L \rangle\): statement List, \(\langle S \rangle\): Statement, \(\langle NLS \rangle\): Non Labeled Statement, \(\langle V \rangle\): Variable, \(\langle E \rangle\): Expression, \(\langle DI \rangle\): DO Instruction, \(\langle O \rangle\): Operand, \(\langle V \rangle\): Identifier

The problem to be tackled here is the determination, in any program, of statements that are inaccessible at execution time, for example, all nonlabeled statements following an unconditional jump. To achieve this, we have to play on the notion of label unconditional jump, and inaccessibility for each statement. Let us therefore define the following attributes:

\[L_{A_S} = 1\quad \text{if the statement } S \text{ is labeled,}\]
\[L_{A_S} = 0\quad \text{otherwise.}\]

\[U_{J_K} = 1\quad \text{if the last statement of } K \text{ is an unconditional jump,}\]
\[U_{J_K} = 0\quad \text{otherwise.}\]
I.IE_K = 0 if the first statement of K is inaccessible to execution (the first statement of a program is assumed to be always accessible).

S.IE_K = 0 if the last statement of K is inaccessible to execution.

We use a notation where each attribute of a nonterminal bears this nonterminal as suffix. On the other hand, it often happens that two attributes, one inherited and the other synthesized, carry information of the same nature. A convenient way of designating them is to use the same mnemonic symbol preceded by S. as “Synthesized” and I. as “Inherited.”

DESCRIPTION (semantic rules).

\[ \langle P \rangle ::= \langle L \rangle ; \]
\[ I.IE_L = 1 \]
\[ \langle L \rangle ::= \langle L' \rangle ; \langle S \rangle \]
\[ UJ_L = UJ_S \]
\[ I.IE_{L'} = I.IE_L \]
\[ S.IE_L = S.IE_S \]
\[ I.IE_S = \text{if } LA_S = 0 \land (UJ_{L'} = 1 \lor S.IE_{L'} = 0) \text{ then } 0 \text{ else } I.IE_L \]

In this production rule the two occurrences of I. are denoted by L and L' to differentiate the respective attributes.

\[ \langle S \rangle ::= \langle NLS \rangle \]
\[ LA_S = 0 \]
\[ UJ_S = UJ_{NLS} \]
\[ I.IE_{NLS} = I.IE_S \]
\[ S.IE_S = S.IE_{NLS} \]
\[ \langle S \rangle ::= \text{Label: } \langle NLS \rangle \]
\[ JA_S = 1 \]
\[ UJ_S = UJ_{NLS} \]
\[ I.IE_{NLS} = I.IE_S \]
\[ S.IE_S = S.IE_{NLS} \]
\[ \langle NLS \rangle ::= \langle DI \rangle ; \langle L \rangle ; \text{END} \]
\[ UJ_{NLS} = 0 \]
\[ I.IE_{DI} = I.IE_{NLS} \]
\[ I.IE_L = I.IE_{NLS} \]
\[ S.IE_{NLS} = I.IE_{NLS} \]
\[ \langle NLS \rangle ::= \text{GOTO Label} \]
\[ UJ_{NLS} = 1 \]
\[ S.IE_{NLS} = I.IE_{NLS} \]
\[ \langle NLS \rangle ::= \langle V \rangle = \langle E \rangle \]
\[ UJ_{NLS} = 0 \]
\[ I.IE_{NLS} = I.IE_{NLS'} \]
\[ \langle DI \rangle ::= \text{DO } \langle V \rangle = \langle E \rangle \text{ TO } \langle E \rangle \text{ BY } \langle E \rangle \]
\[ UJ_{DI} = 0 \]
\[ S.IE_{DI} = I.IE_{DI} \]
These definitions enable the evaluation of all node attributes of a syntactic tree generated by the grammar. The description can in fact be shown to be coherent, i.e., for each syntactic tree there exists an evaluation order \([5, 6]\), forcing the value of an attribute to be calculated only after those on which it depends have been determined (no circularity).

**Example.**

```
GOTO E;
X = 1;
Y = 2;
E: ..............................................
```

Consider the syntactic tree corresponding to these statements:

The previous description gives rise to a system of equations defining node attributes and there exists an evaluation order to compute these attributes, for example:

1. \(I.IE_L = 1\)
2. \(I.IE_{L3} = I.IE_L\)
3. \(I.IE_{L2} = I.IE_{L3}\)
4. \(I.IE_{L1} = I.IE_{L2}\)
5. \(I.IE_{S1} = I.IE_{L1}\)
6. \(S.IE_{S1} = I.IE_{S1}\)
7. \(S.IE_{L1} = S.IE_{S1}\)
8. \(UJ_{S1} = 1\)
9. \(UJ_{L1} = UJ_{S1}\)
10. \(LA_{S0} = 0\)
11. \(I.IE_{S0} = \text{if } LA_{S0} = 0 \land UJ_{L1} = 1 \lor S.IE_{L1} = 0\)
    then 0 else \(I.IE_{L2}\)

etc.

A more precise study of attributes evaluation and in particular of the algorithms determining the coherence of a description is to be found in \([9, 10]\). The hypothesis of coherence of the description is essential for our proof method and will be used throughout this paper.
II. Example of Proof of a Description By Attributes

II.1. Logical Attributes and Associated Axioms

To prove that the preceding description meets the pursued goal implies that for all statement lists of a program as well as for all \( K (K = S, NLS, ID) \) we have:

- If \( IIE_L = 0 \) then the first statement of \( L \) is not accessible to execution.
- If \( SIE_L = 0 \) then the last statement of \( L \) is not accessible to execution.
- If \( IIE_K = 0 \) then statement \( K \) is not accessible.
- If \( SIE_K = 0 \) then statement \( K \) is not accessible.

Each of these assertions depends on a nonterminal. To formalize, we shall introduce the notion of logical attributes. Let us denote the fact that the first or last statement of \( L \) is never executed by \( FNE_L \) and \( LNE_L \), respectively, and the case of a statement \( K \) never executed by \( SNE_K \). Then the assertions to be demonstrated can be written as:

\[
IIE_L = 0 \Rightarrow FNE_L, \quad IIE_K = 0 \Rightarrow SNE_K,
\]

\[
SIE_L = 0 \Rightarrow LNE_L, \quad SIE_K = 0 \Rightarrow SNE_K,
\]

where \( FNE, LNE, \) and \( SNE \) are referred to as logical attributes.

The assertions to be proved thus link up the "calculated" and the logical attributes of the same nonterminal. The logical attributes can be seen as Boolean variables verifying axioms associated with the different production rules as indicated below:

\[
\langle L \rangle ::= \langle S \rangle
\]

\[
FNE_L \leftarrow SNE_S,
\]

\[
SNE_S \leftarrow LNE_L,
\]

\[
\langle L \rangle ::= \langle L' \rangle \langle S \rangle
\]

\[
FNE_L \leftarrow FNE_{L'},
\]

\[
SNE_S \leftarrow LNE_L,
\]

\[
\langle S \rangle ::= \text{label} \langle NLS \rangle
\]

\[
SNE_S \leftarrow SNE_{NLS},
\]

\[
\langle NLS \rangle ::= \langle DL \rangle \langle L \rangle \text{ END}
\]

\[
LA_S = 0 \land U_{J_L} = 1 \Rightarrow SNE_S
\]

\[
LA_S = 0 \land LNE_L' \Rightarrow SNE_S
\]

\[
1 \quad FNE_L \Rightarrow SNE_S
\]

\[
(2) \quad SNE_{NLS} \Rightarrow SNE_{DL}
\]

N.B. Axioms (1) and (2) convey the fact that the semantics of the language allow no entry into an iteration from an external point.

II.2. Induction Demonstration

For all nonterminal \( K (K = S, NLS, ID) \), it has to be shown that

\[ IIE_K = 0 \Rightarrow SNE_K \quad \text{and} \quad SIE_K = 0 \Rightarrow SNE_K. \]
On the other hand, we have to prove

\[ I.IE_L = 0 \Rightarrow FNE_L \quad \text{and} \quad S.IE_L = 0 \Rightarrow LNE_L. \]

For each "semantic rule" defining \( I.IE \) and \( S.IE \), we shall proceed by demonstrating the assertion for the left-hand side attribute of the rule on assuming it to be true for the attributes appearing on the right-hand side. An induction on the evaluation order justifies this approach. It has already been mentioned that the description is coherent, i.e., there exists an evaluation order for attributes in a syntactic tree requiring that a semantic function's argument be computed before the function itself. Evidently, the first calculated attribute has to be independent of all others: In our example this can be indicated by \( I.IE_L = 1 \). Hence, for all nodes of any syntactic tree the justification of an assertion can be given through a finite number of proofs on the nonterminals. Taking the different production rules, the demonstration for the given example would be

\[
\langle P \rangle ::= \langle L \rangle
\]

\(-I.IE_L = 1: \) Here \( I.IE_L = 0 \Rightarrow FNE_L \) is evident.

\[
\langle L \rangle ::= \langle S \rangle
\]

\(-I.IE_S = I.IE_L: \) Assertion to be proved is \( I.IE_S = 0 \Rightarrow SNE_S \). Induction hypothesis is \( I.IE_L = 0 \Rightarrow FNE_L \). The demonstration therefore follows from the equality and the axiom \( FNE_L \Leftrightarrow SNE_S \).

\(-S.IE_S = S.IE_L: \) Assertion to be proved is \( S.IE_L = 0 \Rightarrow LNE_L \). Induction hypothesis is \( S.IE_S = 0 \Rightarrow SNE_S \). The equality and the axiom \( SNE_S \Leftrightarrow LNE_L \) lead to the justification.

\[
\langle L \rangle ::= \langle L' \rangle; \langle S \rangle
\]

\(-I.IE_{L'} = I.IE_L: \) Assertion is \( I.IE_{L'} = 0 \Rightarrow FNE_{L'} \). From the induction hypothesis \( I.IE_L = 0 \Rightarrow FNE_L \) and the axiom \( FNE_L \Leftrightarrow FNE_{L'} \) it can be shown that

\[ I.IE_{L'} = 0 \Rightarrow I.IE_L = 0 \Rightarrow FNE_L \Rightarrow FNE_{L'}. \]

\(-S.IE_S = S.IE_{L'}: \) Assertion is \( S.IE_S = 0 \Rightarrow SNE_S \).

\(-I.IE_S = \text{if } L.A_S = 0 \land (U.J_L' = 1 \lor S.IE_{L'} = 0) \text{ then } 0 \text{ else } I.IE_L: \) Assertion is \( I.IE_S = 0 \Rightarrow SNE_S \).

Case 1. \( L.A_S = 0 \land U.J_L' = 1 \). \( SNE_S \) is deduced from the axiom

\[ L.A_S = 0 \land U.J_L' = 1 \Rightarrow SNE_S. \]

Case 2. \( L.A_S = 0 \land S.IE_{L'} = 0 \). The induction hypothesis \( S.IE_{L'} = 0 \Rightarrow LNE_{L'} \) and the axiom \( L.A_S = 0 \land LNE_{L'} \Rightarrow SNE_S \) lead to \( SNE_S \).

Case 3. \( I.IE_S = I.IE_L \). Induction hypothesis \( I.IE_L = 0 \Rightarrow FNE_L \) and the axiom \( FNE_L \Rightarrow SNE_S \) then allow to conclude that \( I.IE_S = 0 \Rightarrow SNE_S \).
CORRECTNESS OF SYNTAX-DIRECTED PROCESSING

\[ \langle S \rangle ::= \langle NLS \rangle \]

\(-I.IE_{NLS} = I.IE_S\): The assertion \(I.IE_{NLS} = 0 \Rightarrow SNE_{NLS}\) is deduced from the induction hypothesis \(I.IE_S = 0 \Rightarrow SNE_S\) and the axiom \(SNE_S \Rightarrow SNE_{NLS}\).

\(-Similar demonstration for S.IE_S = S.IE_{NLS}\):Assertion is \(S.IE_S = 0 \Rightarrow SNE_S\).

\(\langle S \rangle ::= \text{Label} : \langle NLS \rangle\)

Same demonstration as above.

\(\langle NLS \rangle ::= \text{GOTO Label}\)

\(-S.IE_{NLS} = I.IE_{NLS}:\) Assertion is \(S.IE_{NLS} = 0 \Rightarrow SNE_{NLS}\); induction hypothesis is \(I.IE_{NLS} = 0 \Rightarrow SNE_{NLS}\). The demonstration is then summed up by:

\[ S.IE_{NLS} = 0 \Rightarrow I.IE_{NLS} = 0 \Rightarrow SNE_{NLS} \]

The proofs are quite analogous for the three remaining rules.

III. PROOF METHOD

Let us sum up and generalize the procedure illustrated by the example treated in the preceding section. In addition to “calculated” attributes defined in the description to be proved, logical attributes can be introduced; they verify axioms associated with the different production rules. Each assertion (to be demonstrated) links the attributes (calculated or logical) of the same nonterminal symbol.

In our example, each assertion contained only one calculated attribute so that an induction on the evaluation order of this attribute was straightforward. In general however, an assertion can contain several calculated attributes associated with the same nonterminal and therefore the attribute to which induction is applied should be determined (an induction on the evaluation order of attributes not appearing in the assertions could even be envisaged).

To be more precise, it is agreed that to each assertion linking the attributes of a non-terminal, a calculated attribute of this symbol is to be associated: the leader attribute of the assertion. Since the description is assumed coherent, the induction order on assertions is then the evaluation order of their leader attributes.

The method of proof is then the following: for each production rule \(R\) and each of the related semantic rules: \(A^0_{N_0} = f(A^1_{N_1}, \ldots, A^n_{N_n})\) if \(A^0\) is the leader attribute of an assertion connecting the attributes of the nonterminal \(N_0\), then deduce this assertion from:

\(-\)Assertions relative to the nonterminals \(N_1, \ldots, N_p\) with leader attributes \(A^1, \ldots, A^n\).

\(-\)Semantic rules and axioms relative to the production rule \(R\).

It is this principle that has been applied to the assertions in the preceding example:
This is a page from a document that discusses the equivalence of two attributed descriptions. The page contains mathematical expressions and logical statements. The content is about proving the equivalence of two translation versions, one of which is optimized. This would allow us to conclude that the optimization described by attributes conserves the original semantic content. In a more general sense, this example would demonstrate how our method can be applied to the equivalence proof of descriptions by attributes. The page includes a section on how to treat the local elimination of common redundant subexpressions in a program by the application of "Dependency Number" algorithm to the object code considered to be triples. Each triple consists of a binary operator and two operands which can be constants, variables, or triple numbers. Each triple is referenced by a number giving its position in the sequence of triples. To each triple is associated a dependency number in the following manner:

- dependency number of a constant C is zero, i.e., dep(C) = 0
- initial dependency number of a variable A is zero
- if triple (i) modifies the value of the variable A, then dep(A) = i in the interval |triple (i), next triple modifying A.]
- dependency number of triple (n): opor, opd 1, opd 2 is given by dep(n) = 1 + Max(dep(opd 1), dep(opd 2)).

The following result can then be arrived at: Let (i) and (j) be two triples having the same operator and operands (for a commutative operator the order of operands is immaterial). If i < j and dep(i) = dep(j), then triple (j) is redundant w.r.t. triple (i).

This would allow a certain optimization of the source program when transformed into triples through the elimination of redundant triples. We shall use attributes to define the optimized and the nonoptimized object code and demonstrate their equivalence. This would guarantee, on the one hand, the validity of the optimization algorithm used (i.e., the preceding result) and on the other hand, the correctness of its applications.
CORRECTNESS OF SYNTAX-DIRECTED PROCESSING

The correctness of syntax-directed processing is based on the generation associated with the nonterminal $K$, while the attribute $S.OB_K$ gives the sequence state after this generation. For the optimized program, analogous attributes $I.OBO$ and $S.OBO$, whose values are also triple sequences, are used. These sequences are generated at compile-time and their length depends on the number of triples produced for the nonterminals already processed. For a finite sequence $s$, $s(i)$ denotes the $i$th element and $\text{lg}(s)$ its length.

The dependency number of operands are conveyed by attributes $I.DN$ and $S.DN$ whose values are functions of constants, variables, and triple numbers. Each operand is therefore referenced by an attribute $R$ ($RO$ for the optimized sequence) which gives the address of constants, variables or the number of the last triple of other expression types.

We shall consider the source language defined by the grammar given in Section 1.2, leaving out however, the label, jump, and DO-instruction for the sake of brevity. Below, the two descriptions of the translation into triples are given simultaneously: $S.OB_p$ and $S.OBO_p$ denote, respectively, the nonoptimized and optimized object programs for the source-text $P$:

\[
\langle P \rangle ::= \langle L \rangle \\
I.OB_L = \text{empty sequence} \quad I.OBO_L = \text{empty sequence} \\
S.OB_p = S.OB_L \\
S.OBO_p = S.OBO_L \\
I.DN_L = \text{null function}
\]

\[
\langle L \rangle ::= \langle S \rangle \\
I.OB_S = I.OB_L \\
S.OB_L = S.OB_S \\
S.OBO_L = S.OBO_S \\
I.DN_S = I.DN_L \\
S.DN_L = S.DN_S
\]

\[
\langle L' \rangle ::= \langle L' \rangle; \langle S \rangle \\
I.OB_{L'} = I.OB_L \\
I.OB_S = S.OB_{L'} \\
S.OB_L = S.OB_S \\
S.OBO_L = S.OBO_S \\
I.DN_{L'} = I.DN_L \\
I.DN_S = S.DN_{L'} \\
S.DN_L = S.DN_S
\]

\[
\langle S \rangle ::= \langle V \rangle = \langle E \rangle \\
I.OB_E = I.OB_S \\
I.OBO_E = I.OBO_S
\]
\[ S.OB_S = S.OB_E \| \text{"} = R_E R_V \text{"} \quad S.OBO_S = S.OBO_E \| \text{"} = R_E R_{O_E} \text{"} \]
\[ 1.DN_E = 1.DN_S \]
\[ S.DN_S(x) = S.DN_E(x) \text{ if } x \neq R_V \text{ and } S.DN_S(R_V) = \lg(S.OB_E) + 1 \]

\textit{N.B.} \| \text{ denotes concatenation and } \lg \text{ gives the length of a triple sequence. Redundant assignment triples are not eliminated here: this is a particular case of the elimination of useless assignments [12].}

\[ \langle E \rangle ::= \langle O \rangle \]
\[ S.OB_E = I.OB_E \quad S.OBO_E = I.OBO_E \]
\[ R_E = R_O \quad R_{O_E} = R_{O_O} \]
\[ 1.DN_E = 1.DN_E \]

\[ E ::= E' + O \]
\[ I.OB_{E'} = I.OB_E \quad I.OBO_{E'} = I.OBO_E \]
\[ S.OB_E = S.OB_{E'} \| \text{"} + R_E R_O \text{"} \quad S.OBO_{E'} = \text{if } R_{O_E} = \lg(S.OBO_{E'}) + 1 \text{ then } S.OBO_{E'} \| \text{"} + R_{O_E} R_{O_O} \text{"} \text{ else } S.OBO_{E'} \]
\[ 1.DN_{E'} = 1.DN_E \]
\[ S.DN_E(x) = S.DN_{E'}(x) \text{ if } x \neq R_{O_E} \text{ and } \]
\[ S.DN_E(R_{O_E}) = d \text{ where } \]
\[ d = 1 + \max(S.DN_{E'}(R_{O_E}), S.DN_E(R_{O_O})) \]
\[ R_E = \lg(S.OB_{E'}) + 1 \quad R_{O_E} = \text{if there exists an } i \text{ such that } \]
\[ 1 \leq i \leq \lg(S.OBO_{E'}) \]
\[ \land S.DN_E(i) = d \]
\[ \land (S.OBO_{E'}(i) = \text{"} + R_{O_E} R_{O_O} \text{"} \lor S.OBO_{E'}(i) = \text{"} + R_{O_O} R_{O_E} \text{"}) \]
\[ \text{ then } i \text{ else } \lg(S.OBO_{E'}) + 1 \]

\[ \langle O \rangle ::= \langle V \rangle \]
\[ R_O = R_V \quad R_{O_O} = R_{O_V} \]

\[ \langle O \rangle ::= \text{constant} \]
\[ R_O = \text{locates a constant} \quad R_{O_O} = \text{locates a constant} \]
CORRECTNESS OF SYNTAX-DIRECTED PROCESSING

\[ \langle V \rangle ::= \text{Identifier} \]

\[ R_\nu = \text{locates an identifier} \quad RO_\nu = \text{locates an identifier} \]

IV.3. Assertions to Justify

At this point, we can talk only of equivalence relative to semantics of programs presented in the form of triple sequences. The semantics can be so expressed as to invest the triple sequence with two functions:

- to obtain variable values: Let \( \text{var}(s, x) \) denote the value of the variable or the constant referenced by \( x \) after the execution of the sequence \( s \), for given initial values of the variables.

- to calculate the value of an expression for the case where the last triple is of the form \( \text{''+ab''} \): let \( \text{val}(s) \) denote the value of the expression calculated by the sequence \( s \), for given initial values of the variables.

The functions var and val are defined by induction:

- if \( s \) is the empty sequence, then \( \text{var}(s, x) \) is the initial value of \( x \)

- if \( s \) is the sequence \( s_1 \) followed by the assignment triple \( \text{''=ab''} \), then

\[
\text{var}(s, x) = \text{var}(s_1, x) \quad \text{for } x \neq b,
\]

\[
\text{var}(s, b) = \text{var}(s_1, a), \quad \text{where } a \text{ references a variable or a constant},
\]

\[
\text{var}(s, b) = \text{val}(s_1(1 : a)), \quad \text{where } a \text{ is a triple number}.
\]

- if \( s \) is the sequence \( s_1 \) followed by the triple \( \text{''+ab''} \) (\( a \) references a variable, a constant or can be a triple number, but in the case treated here \( b \) references always a variable or a constant), then

\[
\text{var}(s, x) = \text{var}(s_1, x) \quad \text{for all } x,
\]

\[
\text{val}(s) = \text{var}(s_1, a) + \text{var}(s_1, b) \quad \text{if } a \text{ references a variable or a constant},
\]

\[
\text{val}(s) = \text{val}(s_1(1 : a)) + \text{var}(s_1, b) \quad \text{if } a \text{ is a triple number}.
\]

N.B. \( s_1(1 : a) \) denotes the part of \( s_1 \) that is made up of the first \( a \) triples.

We are now in a position to formulate the assertion to be proved as

\[
\text{var}(S.OB_\nu, x) = \text{var}(S.OBO_\nu, x) \quad \text{for all } x.
\]

Since an induction method has been chosen for the demonstration, we need to show an analogous equivalence for other nonterminals:

(1) \( \text{var}(S.OB_K, x) = \text{var}(S.OBO_K, x) \) for all \( x \), where \( K = P, L, S, E \).

A similar assertion for the associated inherited attribute should also be demonstrated:
(2) \( \text{var}(I.OB_K, x) = \text{var}(I.OBO_K, x) \) whatever the value of \( x \) and for \( K = L, S, E \).
Finally we shall see the necessity of assertions with regards to the expressions:

(3) \( R_E \) triple number \( = R_E = \lg(S.OB_E) \land RO_E \) triple number of \( S.OBO_E \land \text{val}(S.OB_E) = \text{val}(S.OBO_E(1 : RO_E)) \).

(4) \( R_K \) references a constant or a variable \( \Rightarrow R_K = RO_K \) for \( K = E, O, V \).

The leader attributes for these assertions can be \( S.OB_K, I.OB_K, S.OB_E, \) and \( R_K \), respectively. No logical attribute is used here.

IV.4. Proof

The proofs are only given for three production rules; the others are quite straightforward and may safely be left out.

\[ \langle P \rangle ::= \langle L \rangle \]

\(-I.OB_L = \text{empty sequence}: \) Then \( \text{var}(I.OB_L, x) = \text{var}(I.OBO_L, x) \) can be deduced from \( I.OB_L = I.OBO_L = \text{empty sequence} \).

\(-S.OB_p = S.OBO_L: \) Then \( \text{var}(S.OB_p, x) = \text{var}(S.OBO_p, x) \) can be deduced from the semantic rules \( S.OB_p = S.OB_L, S.OBO_p = S.OBO_L \), and the induction hypothesis \( \text{var}(S.OB_L, x) = \text{var}(S.OBO_L, x) \).

\[ \langle S \rangle ::= \langle V \rangle = \langle E \rangle \]

\(-I.OB_E = I.OB_S: \) The assertion to justify is of type (2) with \( K = E \). The demonstration follows immediately.

\(-S.OB_S = S.OB_E \parallel ' = R_E R_V': \) Whatever the value of \( x \), the assertion \( \text{var}(S.OB_S, x) = \text{var}(S.OBO_S, x) \) can be deduced from:

(a) The semantic rules

\[ S.OB_S = S.OB_E \parallel ' = R_E R_V, \]
\[ S.OBO_S = S.OBO_E \parallel ' = RO_E RO_V. \]

(b) The definition of function \( \text{var} \)

\[ \text{var}(S.OB_E \parallel ' = R_E R_V, x) = \text{var}(S.OB_E, x) \quad \text{for} \quad x \neq R_V, \]
\[ \text{var}(S.OB_E \parallel ' = R_E R_V, R_V) = \text{var}(S.OB_E, R_E) \quad \text{if} \quad R_E \text{ references a variable or a constant}, \]
\[ \text{var}(S.OB_E \parallel ' = R_E R_V, R_V) = \text{val}(S.OB_E(1 : R_E)) \quad \text{if} \quad R_E \text{ is a triple number}, \]
\[ \text{var}(S.OBO_E \parallel ' = RO_E RO_V, x) = \text{var}(S.OBO_E, x) \quad \text{for} \quad x \neq RO_V, \]
var(S.OBO_E \parallel '='RO_ERO_V', RO_V) = var(S.OBO_E, RO_E) \quad \text{if } RO_E \text{ references a variable or a constant},

var(S.OBO_E \parallel '='RO_ERO_V', RO_V) = val(S.OBO_E(1 : RO_E)) \quad \text{if } RO_E \text{ is a triple number.}

(c) The induction hypothesis:

\[ \text{var}(S.OB_E, x) = \text{var}(S.OBO_E, x) \quad \text{for all } x, \]
\[ R_E = RO_E \quad \text{if } R_E \text{ references a variable or a constant}, \]
\[ \text{val}(S.OB_E) = \text{val}(S.OBO_E(1 : RO_E)) \quad \text{if } R_E \text{ is a triple number}, \]
\[ R_V = RO_V. \]

\[ \langle E \rangle ::= \langle E' \rangle + \langle O \rangle \]

The demonstration is evident for the assertion of type (2) with \( K = E' \).

The assertion (1): \( \text{var}(S.OB_E, x) = \text{var}(S.OBO_E, x) \) can be deduced from the semantic rules defining \( S.OB_E \) and \( S.OBO_E \), the properties of function \( \text{var} \) and the induction hypothesis (1) applied to \( E' \).

\( RO_E \) is a triple number of \( S.OBO_E \) and \( R_E = \lg(S.OB_E) \) results from the semantic rules.

The assertion \( \text{val}(S.OB_E) = \text{val}(S.OBO_E(1 : RO_E)) \) can be demonstrated for the two situations where:

(a) \( R_E \) is a variable or a constant. Here two cases are distinguished:

Case 1 (the generated triple is not redundant). The relevant semantic rules are \( S.OB_E = S.OB_E' \parallel '+'R_E'RO' \) and \( S.OBO_E = S.OBO_E' \parallel '+'RO'E'RO'O' \); the definition of function \( \text{val} \) yields \( \text{val}(S.OB_E) = \text{var}(S.OBO_E', R_E') + \text{var}(S.OB_E', R_O) \), \( \text{val}(S.OBO_E) = \text{var}(S.OBO_E', RO_E') + \text{var}(S.OBO_E', R_O) \); the induction hypotheses lead to \( \text{var}(S.OB_E', R_E') = \text{var}(S.OBO_E', RO_E') \) and \( R_E' = RO_E' \) as well as to \( \text{var}(S.OB_E', R_O) = \text{var}(S.OBO_E', R_O) \) and \( R_O = R_O \); from which it can be deduced that \( \text{val}(S.OB_E) = \text{val}(S.OBO_E) \). The demonstration is then completed using the fact that \( RO_E = \lg(S.OBO_E) + 1 = \lg(S.OBO_E) \).

Case 2 (the generated triple is redundant: triple \( RO_E = i \)). The relevant semantic rules are \( S.OB_E = S.OB_E' \parallel '+'R_E'RO' \) and \( S.OBO_E = S.OBO_E' \) \; the last triple \( i \) of the sequence \( S.OBO_E(1 : RO_E) \) is \( '+'RO_E', RO'O \) or \( '+'RO'O RO'E' \); the definition of function \( \text{val} \) gives: \( \text{val}(S.OB_E) = \text{var}(S.OBO_E', R_E') + \text{var}(S.OBE_E', R_O) \) and

\[ \text{val}(S.OBO_E(1 : RO_E)) = \text{val}(S.OBO_E(1 : RO_E - 1), RO_E') \]
\[ + \text{var}(S.OBO_E(1 : RO_E - 1), R_O). \]

The induction hypotheses are the same as in the preceding case. If we assume (this
is demonstrated in Section IV.5) that in $S.OBO_{E'}$, no assignment triple to $RO_{E'}$ and $RO_o$ has a number superior to $RO_E = i$, it can be seen that:

$$\text{var}(S.OBO_{E'}(1 : RO_E - 1), RO_{E'}) = \text{var}(S.OBO_{E'}, RO_{E'}) = \text{var}(S.OB_{E'}, R_{E'})$$

and

$$\text{var}(S.OBO_{E'}(1 : RO_E - 1), RO_o) = \text{var}(S.OBO_{E'}, RO_o) = \text{var}(S.OB_{E'}, R_o)$$

thus completing the demonstration.

(b) $R_{E'}$ is a triple number. Similarly two cases are distinguished:

Case 1 (the generated triple is not redundant). The relevant semantic rules are $S.OB_E = S.OB_{E'} \parallel \langle +R_E R_O \rangle$ and $S.OBO_E = S.OBO_{E'} \parallel \langle +RO_{E'} R_O \rangle$. The definition of function $\text{val}$ gives:

$$\text{val}(S.OB_E) = \text{val}(S.OB_{E'}(1 : R_{E'})) + \text{var}(S.OB_{E'}, R_o)$$

and

$$\text{val}(S.OBO_E) = \text{val}(S.OBO_{E'}(1 : R_{E'})) + \text{var}(S.OBO_{E'}, R_o);$$

the induction hypotheses lead to:

$$\text{val}(S.OB_{E'}(1 : R_{E'})) = \text{val}(S.OB_{E'}) = \text{val}(S.OBO_{E'}(1 : R_{E'})),$$

$$\text{var}(S.OB_{E'}, R_o) = \text{var}(S.OBO_{E'}, R_o)$$

and $R_o = RO_o$.

It can thus be deduced that $\text{val}(S.OB_E) = \text{val}(S.OBO_E)$ and the demonstration is then concluded using the fact that $RO_E = \lg(S.OBO_{E'}) + 1 = \lg(S.OBO_E)$.

Case 2 (the generated triple is redundant: triple $RO_E = i$). The relevant semantic rules are $S.OB_E = S.OB_{E'} \parallel \langle +R_E R_O \rangle$ and $S.OBO_E = S.OBO_{E'}$; the triple $(i), \langle +RO_{E'} R_O \rangle$, terminates the sequence $S.OBO_E(1 : R_{E'})$. From the definition of function $\text{val}$, we have:

$$\text{val}(S.OB_E) = \text{val}(S.OB_{E'}(1 : R_{E'})) + \text{var}(S.OB_{E'}, R_o),$$

$$\text{val}(S.OBO_E(1 : R_{E'})) = \text{val}(S.OBO_{E'}(1 : R_{E'})) + \text{var}(S.OBO_{E'}(1 : RO_{E'} - 1), RO_o);$$

the induction hypotheses are the same as in the preceding case. If we assume (cf. Section IV.5) that in $S.OBO_E$ no assignment triple to $RO_o = RO_o$ has a number superior to $RO_E = i$, it can be seen that $\text{var}(S.OB_{E'}, R_o) = \text{var}(S.OBO_{E'}(1 : RO_{E'} - 1), RO_o)$ thus concluding the demonstration.

IV.5. Proof of the Assumed Property

The property assumed in the preceding demonstration, namely, that in the sequence $S.OBO_{E'}$ no assignment triple to $RO_{E'}$ and $RO_o$ has a number superior to $RO_E = i$, has to be justified.

To do this, it would be sufficient to demonstrate the following properties of dependency numbers:

- if $x$ is a triple number, then $S.DN_{E'}(x) \leq x$

- if $x$ locates a variable or a constant, then $S.DN_{E'}(x)$ gives the number of the last assignment triple to $x$ in $S.OBO_{E'}$ if it exists, otherwise $S.DN_{E'}(x) = 0$. 
In fact, referring to the semantic rules relative to \( \langle E \rangle ::= \langle E' \rangle + \langle O \rangle \), the first property would imply
\[
i \geq S.DN_E(i) = d > S.DN_E(RO_E),
\]
\[
i \geq S.DN_E(i) - d > S.DN_E(RO_o).
\]
The last assignment triple to \( RO_{E'} \) and \( RO_O \), if such a triple exists, would have a number inferior to \( i \).

To give a demonstration by induction analogous assertions for \( S.DN_L \), \( S.DN_S \), as well as for the associated inherited attributes would have to be proved. More precisely, let us prove for \( K = L, S, E \):

(5) If \( x \) is a triple number, then \( I.DN_K(x) \leq x \) and if \( x \) locates a variable or a constant, then \( I.DN_K(x) \) is the number of the last assignment triple to \( x \) in \( I.OBO_K \) if it exists, otherwise \( I.DN_K(x) = 0 \).

(6) If \( x \) is a triple number, then \( S.DN_K(x) \leq x \) and if \( x \) locates a variable or a constant, then \( S.DN_K(x) \) is the number of the last assignment triple to \( x \) in \( S.OBO_K \) if it exists, otherwise \( S.DN_K(x) = 0 \).

(5) and (6) are associated with \( I.DN_K \) and \( S.DN_K \) attributes, respectively. The demonstration is given only for two production rules, it is straightforward for the others.

\[
\langle S \rangle ::= \langle V \rangle = \langle E \rangle
\]

\(-I.DN_E = I.DN_S\): The assertion to justify is of type (5) for \( K = E \); the demonstration is evident.

\(-S.DN_S(x) = S.DN_E(x) \) for \( x \neq RO_V \) and \( S.DN_S(RO_V) = \log(S.OBO_E) + 1 \). The assertion (6) for \( K = S \) and \( x \neq RO_V \) can be deduced from the induction hypothesis (6) for \( K = E \) and the semantic rule \( S.DN_S(x) = S.DN_E(x) \) for \( x = RO_V \), the assertion is verified since the number of the last triple affecting \( RO_V \) in \( S.OBO_S = S.OBO_E \) \('=RO_ERO_V'\) is \( \log(S.OBO_E) + 1 \).

\[
\langle E \rangle ::= \langle E' \rangle + \langle O \rangle
\]

\(-I.OB_F = I.OB_E\): The assertion to justify is of type (5) for \( K = E' \); the demonstration is evident.

\(-S.DN_E(x) = S.DN_E'(x) \) for \( x \neq RO_E \) and \( S.DN_E(RO_E) = 1 + \max(S.DN_{E'}, (RO_E), S.DN_F(RO_O)) \): The assertion (6) for \( K = E \) and \( x \neq RO_E \) is deduced from the induction hypothesis (6) for \( K = E' \) and the semantic rule \( S.DN_E(x) = S.DN_E'(x) \); for \( x = RO_E \), the assertion has to be justified in the following two cases:

Case 1. \( RO_E = i \) such that \( S.DN_E(i) - d = S.DN_E(RO_E) \). From the induction hypothesis \( S.DN_E(i) \leq i \), hence \( S.DN_E(RO_E) \leq RO_E \).
Case 2. \( RO_E = \lg(S.OBO_E) + 1 \). From the induction hypothesis, \( S.DN_E(RO_O) \)
is either zero or equal to an assignment triple number in \( S.OBO_E \), and \( S.DN_E(RO'_E) \)
is either inferior to the number of \( RO'_E \) in \( S.OBO_E \) or is zero or is an assignment triple
of \( S.OBO_E' \). In all cases \( S.DN_E(RO_E) = d < 1 + \lg(S.OBO_E) = RO_E \).

V. SYNTHESIZED INDUCTION

In certain cases, attribute properties can be demonstrated in a simpler way at all
nodes of a syntactic tree by choosing a bottom up approach: the proof would then consist
of showing that the properties of a node can be derived from those of its "sons." This
method, referred to as synthesized induction, can be outlined as follows:

For each production rule \( R \), deduce the assertions related to the left-hand side non-
terminal of \( R \) from:

- The assertions relative to the right-hand side nonterminals.
- The semantic rules and axioms relative to \( R \).

It is clear that synthesized induction is simply a special case of the general proof
principle given in Section III. It would be sufficient to introduce a synthesized attribute \( S \),
such that for each production rule \( R \):

\[
\langle N \rangle ::= \langle N_1 \rangle \cdots \langle N_n \rangle \text{ there exists a definition } S_N = f(S_{N_1} \cdots S_{N_n}).
\]

The attribute \( S \) is then associated with each assertion that has to be proved.

Proof by synthesized induction does not apply to the preceding examples. But if
we reconsider the example of Section IV with the view of demonstrating that the attribute
description corresponds exactly to one's intuitive aim, in other words, that for all
statements \( K, I.OB_K = S.OB_K(1 : \lg(I.OB_K)) \) is true, then we can use the synthesized
induction method.

VI. CONCLUSION

The formalism of attributes applied to the definition of language processing systems
seems in addition to imparting greater clarity to the process, to offer facilities for the
verification of the processing described, particularly useful in compiler writing and
compiler correctness proofs.

A method has been given to prove that an attributed description effectively carries
out its assigned role and the proof of semantic equivalence outlined in this paper could
open the way for similar equivalence proofs in program transformations in general.

Moreover, the method described has the advantage of being modular since the proof
is subdivided into a certain number of independent demonstrations, each one based
on the semantic rules and the axioms associated with a particular production rule.
The semantics is descriptive rather than algorithmic and independent of any parsing scheme. This confers flexibility on the evaluation order of attributes and the correctness proof method does not have to be confined to bottom-up induction approach.

REFERENCES

13. D. NEEL AND M. AMIRCHAHY, Semantic attributes and improvement of generated code, in "ACM 74, November 11-13, San Diego, California, 1974."
14. D. NEEL, M. AMIRCHAHY, AND M. MAZAUD, Optimization of generated code by means of attributes: Local elimination of common redundant sub-expressions, in "GI 74, October 9-12, Berlin, Germany, 1974."
15. H. GANZINGER, Deriving proof rules for programming language constructs from static and dynamic attribute structures, in "Conference on Theoretical Computer Science, Waterloo, August 1977."