ISSN 0105-8517

# ATTRIBUTE GRAMMARS

# AND

# MATHEMATICAL SEMANTICS

by

Brian H. Mayoh

DAIMI PB-90

August 1978

Computer Science Department
**AARHUS UNIVERSITY**
Ny Munkegade – DK 8000 Aarhus C – DENMARK
*Telephone: 06 – 12 83 55*

# Attribute Grammars and Mathematical Semantics

Brian H. Mayoh
Department of Computer Science
University of Aarhus
Aarhus, Denmark

Keyword: mathematical semantics, attribute grammars, compiler properties.

## Abstract

Attribute grammars and mathematical semantics are rival language definition methods. We show that any attribute grammar G has a reformulation MS(G) within mathematical semantics. Most attribute grammars have properties that discipline the sets of equations the grammar gives to derivation trees. We list six such properties, and show that for a grammar G with one of these properties both MS(G) and the compiler for G can be simplified. Because these compiler friendly properties are of independent interest, the paper is written in such a way that the first and last sections do not depend on the other sections.



Keywords: mathematical semantics, attribute grammars, compiler properties.


## 1. Introduction

Attribute grammars [9] give a systematic way of expressing such restrictions on a programming language as:
variables must be declared before use, the types of the two sides of an assignment statement must agree. In this paper we show that any attribute grammar can be given an equivalent and elegant formulation within the mathematical semantics of D. Scott and C. Strachey, [16, 17]. This reformulation is of interest because of the widespread acceptance of the advantages of mathematical semantics for the description of real programming languages [1, 2, 5, 12, 13, 18, 19].

Before looking at the details of the reformulation let us look at Knuth's simple example of an attribute grammar BIN for binary notation:

| | |
|---|---|
| $B \to 0$ | $v[B] = 0$ |
| $B \to 1$ | $v[B] = 2^{c[B]}$ |
| $L \to B$ | $v[L] = v[B],\ c[B] = c[L],\ l[L] = 1$ |
| $L_0 \to L_1 B_2$ | $v[L_0] = v[L_1] + v[B_2],\ c[B_2],\ c[L_0]$ |
| | $c[L_1] = c[L_0] + 1,\ l[L_0] = l[L_1] + 1$ |
| $N \to L$ | $v[N] = v[L],\ c[L] = 0$ |
| $N \to L_1 \cdot L_2$ | $v[N] = v[L_1] + v[L_2],\ c[L_1] = 0,$ |
| | $c[L_2] = -l[L_2]$ |

As explained in [9, p. 131] one can deduce from the grammar that the number 13.25 is the meaning of the expression 1101.01, because the equations given by the grammar can be ordered suitably. This difficulty with the ordering of equations does not arise when the grammar is reformulated within mathematical semantics:

$$\begin{array}{lll} bv[0](c) & = 0 & \\ bv[1](c) & = 2^{c} & \\ lv[B](c) & = bv[B](c) & ll[B] = 1 \\ lv[LB](c) & = lv[L](c+1) + bv[B]c & ll[LB] = ll[L] + 1 \\ nv[L] & = lv[L](0) & \\ nv[L_1.L_2] & = lv[L_1](0) + lv[L_2](c_2) \text{ where } c_2 = -ll[L_2] & \end{array}$$

Here we have the definition of four functions : bv for the synthesized attribute v of the symbol B, lv for the synthesized attribute v of the symbol L, ll for the synthesized attribute l of the symbol L,and nv for the synthesized attribute v of the symbol N. The first two functions have an argument in round brackets for an inherited attribute. All four functions have an argument in square brackets for derivation trees. In all our examples we will have an unambiguous grammar so we can use strings instead of trees for square bracket arguments. The deduction that the number 13.25 is the meaning of the string 1101.01 now becomes :

$$\begin{aligned} nv[1101.01] &= lv[1101]0 + lv[01]c_2 \text{ where } c_2 = -ll[01] \\ &= lv[1101]0 + lv[01](-2) \\ &= (lv[110]1 + 1) + (lv[0](-1) + 0.25) \\ &= (12 + 1) + (0 + 0.25) = 13.25 \end{aligned}$$

This example is too small to justify the claim - the reformulation of an attribute grammar within mathematical semantics is easier to understand because it only uses functions, whereas an attribute grammar uses attributes, functions and equations. The example in section 4 better illustrates the advantages of a reformulation MS(G) within mathematical semantics of an attribute grammar G. There is always an MS(G), equivalent to G (theorem 1); if G is well defined, then MS(G) does not use recursion (theorem 2). Some attribute grammars have properties, that discipline the set of equations the grammar gives to derivation trees. If a grammar G has one of these properties then the following table shows that both MS(G) and the compiler for G can be simplified.

| Property | Compiler Simplification | MS(G) Simplification |
|---|---|---|
| unordered | subtrees in arbitrary order | as compiler |
| ordered | subtrees from left to right | as compiler |
| reordered | subtrees in fixed order | as compiler |
| tangled | one pass | no splitting |
| benign | attributes in fixed order | determinate |
| well defined | - | no recursion |

Because these compiler friendly properties are of independent interest this paper has been written in such a way that those unconcerned with mathematical semantics can omit all but the last section, and scan the earlier sections when they meet undefined notation.

## 2. Reformulation of an arbitrary attribute grammar.

An attribute structure consists of:

(1) disjoint sets $\mathfrak{a}$, $\underline{A}$, $\overline{A}$,

(2) for each $X$ in $\mathfrak{a}$, subsets $\underline{X} \subset \underline{A}$, and $\overline{X} \subset \overline{A}$;

(3) for each $a$ in $\overline{A} \cup \underline{A}$, a set $V^{o}_{a}$.

The elements of $\mathfrak{a}$ are called symbols, the elements of $\underline{A}$ are called synthesized attributes, and the elements of $\overline{A}$ are called inherited attributes.

For each $X$ in $\mathfrak{a}$ we define

$SYN^{o}[X]$, the Cartesian product of $V^{o}_{a}$ for $a$ in $\underline{X}$

$INH^{o}[X]$, the Cartesian product of $V^{o}_{a}$ for $a$ in $\overline{X}$.

By convention $SYN^{o}[X]$ ($INH^{o}[X]$) has precisely one element if $\underline{X}$ ($\overline{X}$) is empty.
An attribute grammar consists of:

(1) a context free grammar $(\mathfrak{T}, \mathfrak{N}, S, \mathfrak{P})$ ; where the start symbol S does not occur on the right side of a production;

(2) an attribute structure such that $\mathfrak{A} = \mathfrak{T} \cup \mathfrak{N}$, $\overline{S}$ is empty, and $\underline{X}$ is empty for X in $\mathfrak{T}$;

(3) for every production $X_{p,0} \to X_{p,1} \dots X_{p,-1}$ in $\mathfrak{P}$ we have a partial function $f_p^o : L_p^o \to (R_p^o \to R_p^o)$

$$L_p^o = INH^o(X_{p,0}) \times SYN^o(X_{p,1}) \times SYN^o(X_{p,2}) \times \dots \times SYN^o(X_{p,-1})$$
$$R_p^o = SYN^o(X_{p,0}) \times INH^o(X_{p,1}) \times INH^o(X_{p,2}) \times \dots \times INH^o(X_{p,-1})$$

Here and later we avoid a sea of subscripts by using a convention due to B. Rosen in which $X_{p,-1}$, rather than $X_{p,n_p-1}$, is the last symbol of production p. In practice we usually have a function $q_p^o : L_p^o \to R_p^o$ such that $f_p^o(l)(r) = q_p^o(l)$ for all l in $L_p^o$ and r in $R_p^o$, and we say that an attribute grammar is in normal form if we have such a function for each production.

<u>Example</u>

For the attribute grammar BIN we have :

$\underline{B} = \{v\}$ $\overline{B} = \{c\}$ $\underline{L} = \{v, l\}$ $\overline{L} = \{c\}$ $\underline{N} = \{v\}$ $\overline{N} = \{\ \}$

$SYN^o(B) = V_v^o$ $SYN^o(L) = V_v^o \times V_l^o$ $SYN^o(N) = V_v^o$

$INH^o(B) = V_c^o$ $INH^o(L) = V_c^o$ $INH^o(N) = \{\cdot\}$ .

| Syntactic rule | Semantic rule |
|---|---|
| $B \to 0$ | $f_a^o : V_c^o \to (V_v^o \to V_v^o)$ |
| | $f_a^o\ (c[B])(v[B]) = 0$ |
| $B \to 1$ | $f_b^o : V_c^o \to (V_v^o \to V_v^o)$ |
| | $f_b^o\ (c[B])(v[B]) = 2^{c[B]}$ |
| $L \to B$ | $f_c^o : V_c^o \times V_c^o \to (V_v^o \times V_l^o \times V_c^o \to V_v^o \times V_l^o \times V_c^o)$ |
| | $f_c^o\ (c[L],\ v[B])(v[L]),\ l[l],\ c[B]) = (v[B],\ 1,\ c[L])$ |

$L_0 \to L_1 B_2$ $\qquad f_d^o : V_c^o \times V_v^o \times V_l^o \times V_v^o \to (V_v^o \times V_l^o \times V_c^o \times V_c^o \to V_v^o \times V_l^o \times V_c^o \times V_c^o)$

$$f_d^o\,(c[L_0], v[L_1], l[L_1], b[B_2])(v[L_0], l[L_0], c[L_1], c[B_2])$$
$$= (v[L_1] + v[B_2],\ l[L_1] + 1,\ c[L_0] + 1,\ c[L_0])$$

$N \to L$ $\qquad f_e^o : V_v^o \times V_l^o \to (V_v^o \times V_c^o \to V_v^o \times V_c^o)$

$$f_e^o\ (v[L],\ l[L])\ (v[N],\ c[L]) = (v[L],\ 0)$$

$N \to L_1 \cdot L_2$ $\qquad f_f^o : V_v^o \times V_l^o \times V_v^o \times V_l^o \to (V_v^o \times V_c^o \times V_c^o \to V_v^o \times V_c^o \times V_c^o)$

$$f_f^o\ (v[L_1], l[L_1], v[L_2], l[L_2])(v[N], c[L_1], c[L_2])$$
$$= (v[L_1] + v[L_2],\ 0,\ -l[L_2])$$

Note that our reformulation of BIN borrows notation like $v[B]$ for attribute values from the original formulation, and it shows that the attribute grammar is in normal form. The differences between our definition of attribute grammar and that in [9] are minor and inessential, but they pave the way to the lattices and functions of mathematical semantics. For each symbol X in $\mathfrak{N} \cup \mathfrak{T}$ the productions of the grammar give $\mathrm{DOM}^o(X)$, the set of derivation trees that can be generated from X. As described in [16, 17] one can convert the sets $V_a^o$, $\mathrm{SYN}^o(X)$, $\mathrm{INH}^o(X)$ $\mathrm{DOM}(X)$ by adding a bottom element $\perp$ and a top element T, to lattices $V_a$, SYN(X), INH(X), DOM(X), and one can form a lattice of continuous functions

$$\mathrm{CONT}(X) = \mathrm{DOM}(X) \to (\mathrm{INH}(X) \to \mathrm{SYN}(X)).$$

A reformulation of a grammar in mathematical semantics will define precisely one element of CONT(S).

Convention

When specifying a function x over DOM(X), we may do so by a set of equations

$$x[X_{p,1} \ldots X_{p,-1}] = \ldots .$$

with one equation for each production p with $X_{p,0} = X$.

We use $x[X_{p,1} \ldots X_{p,-1}]$ as a convenient way of writing: the value of x on a derivation tree of the form

$$\begin{tikzcd} & X_{p,0} \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\ \mathfrak{T}_{p,1} & \mathfrak{T}_{p,2} \; - - - - - & \mathfrak{T}_{p,-1} \end{tikzcd}$$

where $\mathfrak{T}_{p,1} \dots \mathfrak{T}_{p,-1}$ are derivation trees with $X_{p,1} \dots X_{p,-1}$ at their roots. Because DOM(X) is the lattice sum of $\mathrm{DOM}(X_{p,1}) \times \dots \times \mathrm{DOM}(X_{p,-1})$ for p such that $X_{p,0} = X$, our sets of equations do determine functions over DOM(X). The equation pairs for bv, ll, lv, nv in section 1 determine functions

$$\begin{array}{lll} bv & : & \mathrm{DOM}(B) \to V_c \to V_v \\ ll & : & \mathrm{DOM}(L) \to V_l \\ lv & : & \mathrm{DOM}(L) \to V_c \to V_v \\ nv & : & \mathrm{DOM}(N) \to V_v \end{array}$$

When specifying these functions we used the convention: parentheses can be omitted if this does not lead to confusion. This convention usually allows us to omit parentheses around empty sets of arguments.

<u>Definition 1</u> Let $G = (\mathfrak{T}, \mathfrak{N}, S, \mathfrak{P})$ be an attribute grammar.
An <u>assignment</u> to a derivation tree $\pi$ of G is a pair of functions (sy, in) from nodes of $\pi$ to attribute values such that:

$$sy(u) \in \mathrm{SYN}(X_u) \;\&\; in(u) \in \mathrm{INH}(X_u)$$

where $X_u$ is the symbol at node u. The assignment is said to be <u>complete</u> if for all nodes u we have

$$sy(u) \in \mathrm{SYN}^{o}(X_u) \;\&\; in(u) \in \mathrm{INH}^{o}(X_u).$$

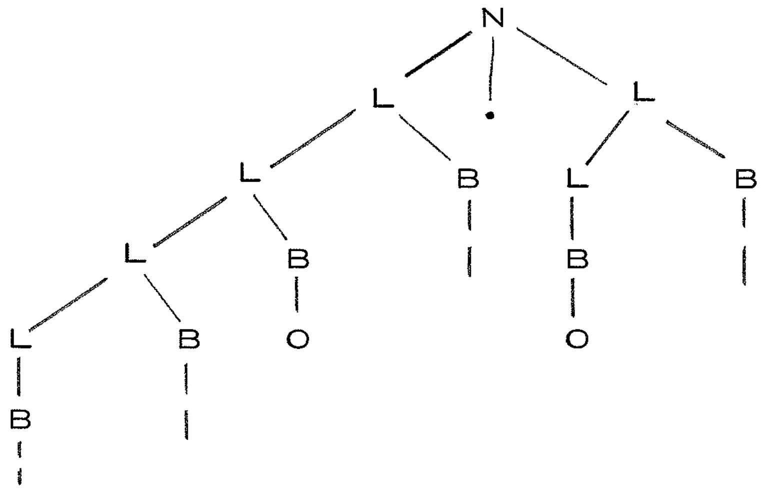
For every complete assignment (sy, in) we can define Next(sy, in) as the assignment (sy', in') given by

$$(*) \quad (sy'(u_0), in'(u_1) \dots in'(u_{-1})) =$$
$$= f_p^{o}(in(u_0), sy(u_1) \dots sy(u_{-1}))\,(sy(u_0), in(u_1) \dots in(u_{-1}))$$

for each application $u_0 \to u_1 \ldots u_{-1}$ of the production $X_{p,0} \to X_{p,1} \ldots X_{p,-1}$ in the tree $\pi$.

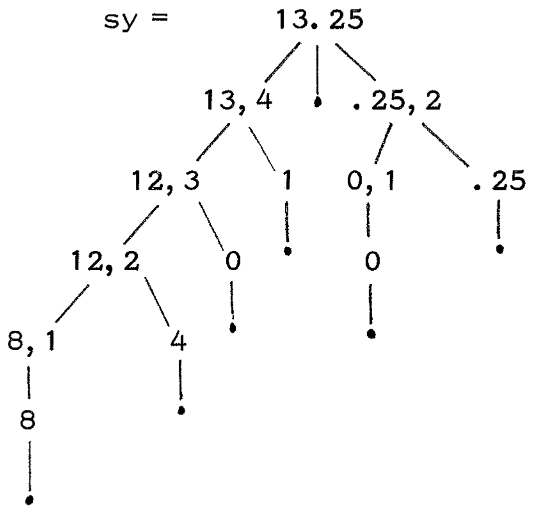
The assignment (sy, in) <u>fits</u> $\pi$ if (sy, in) = Next(sy, in). <u>The grammar G assigns w to $\pi$</u> if w = sy (root of $\pi$) for every complete assignment (sy, in) that fits $\pi$.

<u>Example</u> The derivation tree

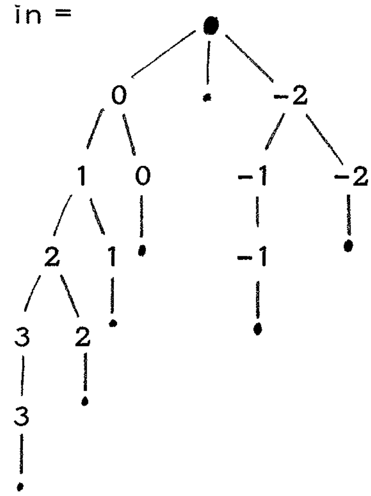



for the grammar BIN fits the assignment

sy =




in =



Def. 1 Ctd.

An assignment $\tau$ to a derivation tree $\pi$ gives a value $\tau(u,\alpha)$ to each attribute $\alpha$ of each node u. We say $\pi$ has a computation sequence if there is a sequence $(u_1,\alpha_1) \ldots (u_n,\alpha_n)$ such that

(1) each $u_j$ is a node of $\pi$ ;

(2) each $\alpha_j$ is an attribute of the symbol at the node $u_j$ ;

(3) the pair $(u,\alpha)$ occurs in the sequence for each attribute a of each node u ;

(4) if $\tau$ and $\tau'$ are complete assginments such that

$$\tau(u_1,\alpha_1) = \tau'(u_1,\alpha_1) \ldots \tau(u_{j-1},\alpha_{j-1}) = \tau'(u_{j-1},\alpha_{j-1})$$

then $\mathrm{Next}(\tau)(u_j,\alpha_j) = \mathrm{Next}(\tau')(u_j,\alpha_j) \neq \perp$ .

Lemma If the derivation tree $\pi$ has a computation sequence, then there is precisely one complete assignment that fits $\pi$.

Proof We define an assignment $\tau_W$ by :

if $\tau$ is any complete assignment such that

$$\tau(u_1,\alpha_1) = \tau_W(u_1,\alpha_1) \ldots \tau(u_{j-1},\alpha_{j-1}) = \tau_W(u_{j-1},\alpha_{j-1})$$

then $\tau_W(u_j,\alpha_j) = \mathrm{Next}(\tau)(u_j,\alpha_j)$.

A simple induction argument using requirement (4) in the definition of computation sequence gives

$$\tau_W(u_j,\alpha_j) \neq \perp \text{ does not depend on the choice of } \tau$$

This implies that $\tau_W$ is a complete assignment.

If we take $\tau_W$ as $\tau$ in the definition of $\tau_W(u_j,\alpha_j)$, we get $\tau_W(u_j,\alpha_j) = \mathrm{Next}(\tau_W)(u_j,\alpha_j)$ so the complete assignment $\tau_0$ fits the tree $\pi$. Suppose $\tau_1$ is a complete assignment that fits $\pi$. If we have $\tau_W(u_1,\alpha_1) = \tau_1(u_1,\alpha_1) \ldots \tau_W(u_{j-1},\alpha_{j-1}) = \tau_1(u_{j-1},\alpha_{j-1})$ requirement (4) gives $\mathrm{Next}(\tau_W)(u_j,\alpha_j) = \mathrm{Next}(\tau_1)(u_j,\alpha_j)$ and $\tau_W(u_j,\alpha_j) = \tau_1(u_j,\alpha_j)$ follows from $\tau_W = \mathrm{Next}(\tau_W)$ & $\tau_1 = \mathrm{Next}(\tau_1)$.

We infer that $\tau_W = \tau_1$.

Theorem 1 For any attribute grammar G with start symbol S we can define a function s in CONT(S) such that for any derivation tree $\pi$ we have

(a) if $s[\pi] = w \in SYN^{o}(s)$, then G assigns w to $\pi$ ;

(b) if $\pi$ has a computation sequence and G assigns w to $\pi$, then $s[\pi] = w$.

Proof

(a) If we extend the functions $f_p^o$ to continuous functions $f_p : L_p \to (R_p \to R_p)$, and we use $f_p$ instead of $f_p^o$ in the equations (*) in the definition, our function Next becomes a continuous function from assignments to assignments. For any derivation tree $\pi$ there is a least assignment $\tau$ satisfying $\tau = \text{Next}(\tau)$. If (sy, in) is this least assignment and we take sy(root) as the value of $s[\pi]$, then "(sy, in) is less than every assignment that fits $\pi$" gives part (a) of our theorem.

(b) Let us agree on the following continuous extension of $f_p^o$ :

$$f_p(l)(r) = \text{greatest lower bound of } f_p^o(l')(r') \text{ for } l \subset l',\ r \subset r'$$

and look at the definition of $\tau_w$ in the proof of the lemma. Because of the way we have extended $f_p^o$ we have

$$\tau_w(u_j, \alpha_j) \;=\; \text{Next}(\tau)(u_j, \alpha_j)$$

for every assignment satisfying

$$\tau_w(u_1, \alpha_1) = \tau(u_1, \alpha_1) \;\ldots\; \tau_w(u_{j-1}, \alpha_{j-1}) = \tau(u_{j-1}, \alpha_{j-1}).$$

Suppose we define $\tau_j = \text{Next}(\tau_{j-1})$ and take $\tau_0$ as the assignment that gives $\perp$ to all attributes of all nodes in a derivation tree. If we have

$$\tau_w(u_1, \alpha_1) = \tau_{j-1}(u_j, \alpha_j) \;\ldots\; \tau_w(u_{j-1}, \alpha_{j-1}) = \tau_{j-1}(u_{j-1}, \alpha_{j-1})$$

we also have $\tau_w(u_j, \alpha_j) = \text{Next}(\tau_{j-1})(u_j, \alpha_j) = \tau_j(u_j, \alpha_j)$.

Since Next is continuous we also have

$$\tau_w(u_1, \alpha_1) = \tau_j(u_1, \alpha_1) \;\ldots\; \tau_w(u_{j-1}, \alpha_{j-1}) = \tau_j(u_{j-1}, \alpha_{j-1})$$

and induction gives

$$\tau_w(u_1,\alpha_1) = \tau_1(u_1,\alpha_1) \ \dots \ \tau_w(u_r,\alpha_r) = \tau_r(u_r,\alpha_r)$$

so $\tau_w$ agrees with the least assignment $\tau_0 \cup \tau_1 \cup \dots$

If the grammar G assigns w to $\pi$, then the lemma ensures that w is the value of the synthesized attributes in the complete assignment $\tau_w$. By definition $s[\pi]$ is the value of these attributes in the least assignment. These two values must be the same.

Comment So far we have only considered assignments to derivation trees with the start symbol of an attribute grammar at their roots. For any derivation tree $\pi$ with root symbol $X \in \mathfrak{N} \cup \mathfrak{T}$ we can extend definition 1 and the proof of theorem 1 to give a function $x[\pi]$ in $\mathrm{INH}(X) \rightarrow \mathrm{SYN}(X)$. These functions are defined by the equations

$$sy_0 = x_{p,0}[X_{p,1} \dots X_{p,-1}]\, in_0$$

$$sy_1 = x_{p,1}[X_{p,1}]\, in_1 \ \dots \ sy_{-1} = x_{p,-1}[X_{p,-1}]\, in_{-1}$$

$$(sy_0, in_0, \dots in_{-1}) = f_p(in_0, sy_1 \ \dots \ sy_{-1})(sy_0, in_1 \ \dots \ in_{-1})$$

This was proved in an earlier version of this paper but the details are so similar to those for the independent result in [4] that they are omitted here. In a suitable specification language, the unique function $x_{p,0}$ given by equations is :

$$(**) \quad x_{p,0}[X_{p,1} \ \dots \ X_{p,-1}]\, in_0 = YH \downarrow 1$$

$$\underline{\text{where}}\ H(sy_0, in_1 \ \dots \ in_{-1})$$

$$= f_p(in_0, x_{p,1}[X_{p,1}]\, in_1, \ \dots, \ x_{p,-1}[X_{p,-1}]\, in_{-1})\,(sy_0, in_1 \dots in_{-1})$$

Here Y is the fix point operator, $\downarrow 1$ selects the first component of a list, and we include the trivial functions $x_{p,i}$ for terminal symbols $X_{p,i}$. In practice such trivial functions can be omitted.

Note that $x_{p,0}$ is a member of CONT($X_{p,0}$) and s in theorem 1 is the least upper bound in CONT(S) of the functions $x_{p,0}$ for the productions with $S = X_{p,0}$.

Applying our construction to the grammar BIN gives the somewhat obscure

$$
\begin{array}{lll}
b[0]c & = & YH \downarrow 1 \ \ \underline{\text{where}}\ H(v) = 0 \\
b[1]c & = & YH \downarrow 1 \ \ \underline{\text{where}}\ H(v) = 2^c \\
l[B]c & = & YH \downarrow 1 \ \ \underline{\text{where}}\ H((v, l), in_1) = ((b[B]in_1, l), c) \\
l[LB]c & = & YH \downarrow 1 \ \ \underline{\text{where}}\ H((v, l), in_1, in_2) \\
 & & \qquad = ((l[L]in_1 \downarrow 1 + b[B]in_2,\ l[L]in_1 \downarrow 2+1, c+1, c) \\
n[L] & = & YH \downarrow 1 \ \ \underline{\text{where}}\ H(v, in_1) = (l[L]in_1 \downarrow 1,\ 0) \\
n[L_1.L_2] & = & YH \downarrow 1 \ \ \underline{\text{where}}\ H(v, in_1, in_2) \\
 & & \qquad = (l[L_1]in_1 \downarrow 1 + l[L_2] \downarrow 1,\ 0,\ -l[L_2]in_2 \downarrow 2)
\end{array}
$$

Straightforward fixed point elimination gives :

$$
\begin{array}{lll}
b[0]c & = & 0 \\
b[1]c & = & 2^c \\
l[B]c & = & (b[B]c,\ 1) \\
l[LB]c & = & (v_1 + b[B]c,\ l_1 + 1)\ \underline{\text{where}}\ (v_1, l_1) = l[L](c+1) \\
n[L] & = & l[L]0 \downarrow 1 \\
n[L_1.L_2] & = & v_1 + v_2\ \ \underline{\text{where}}\ \ (v_1, l_1) = l[L_1]0 \\
 & & \qquad\qquad \underline{\text{and}}\ \ (v_2, l_2) = l[L_2](-l_2)
\end{array}
$$

Replacing the last line by :

$$\underline{\text{and}}\ (v_2, l_2) = YH\ \text{where}\ H(v, l) = l[L_2](-l)$$

makes the recursion explicit.

## 3 Reformulation of a well defined attribute grammar

In this section we show that recursion is not needed when a well defined attribute grammar is reformulated within mathematical semantics.

Definition 2 An attribute grammar $G = (\mathfrak{N}, \mathfrak{T}, S, \mathfrak{P})$ is well defined, if the test in [9] shows G is not circular.

Theorem 2 For any well defined attribute grammar G with start symbol S we can define a function s in CONT(S) such that for any derivation tree $\pi$ we have

(a) the specification of s does not use recursion or the fix point operator Y ;

(b) G assigns w to $\pi \Leftrightarrow s[\pi] = w$ ;

(c) $s[\pi] \neq \perp$ .

Proof

The algorithm for testing whether an attribute grammar is well defined [9, correction] generates a finite set of directed graphs. These graphs are of three kinds. For each element X in $\mathfrak{N} \cup \mathfrak{T}$ we have a set of symbol graphs SYM(X) showing how the synthesized attributes may depend on the inherited attributes of X. For each production $X_{p,0} \rightarrow X_{p,1} \ldots X_{p,-1}$ we have:

(1) a production graph $D_p$ with arrows to the synthesized attributes of $X_{p,0}$ and the inherited attributes of $X_{p,1} \ldots X_{p,-1}$ from the zero, one or more attributes they depend upon;

(2) a set COMP(p) of composite graphs of the form $D_p[Q(1) \ldots Q(-1)]$ where $Q(1) \in SYM(X_{p,1}) \ldots Q(-1) \in SYM(X_{p,-1})$.

Knuth's test for circularity generates the composite graphs and SYM(X) for X in $\mathfrak{N}$ from the production graphs and SYM(X) for X in $\mathfrak{T}$ . If any composite graph contains a cycle, then our attribute grammar G is not well defined; otherwise these graphs tell us how to replace (∗∗) in the proof of theorem 1 by a specification with no implicit or explicit recursion. As we shall see in the next section this reformulation is particularly simple if for every p the union of the graphs in COMP(p) contains no cycle. Even although this simplification is advocated in [4] and almost always possible in practice, we have to treat the general case if we are to prove the theorem. The non-determinism that plagues very general attribute grammars then enters in the form of joins in function lattices. For each symbol X in $\mathfrak{n} \cup \mathfrak{T}$, for each graph $\Gamma$ in SYM(X), and each synthesized attribute $\alpha$ in $\underline{X}$, we introduce a function

$$\text{symbol}\ (\Gamma, \alpha) : \text{DOM}(X) \to W(\Gamma, \alpha) \to V_\alpha$$

where $W(\Gamma,\alpha)$ is the subset of INH(X) given by the arrows going to the node for $\alpha$ in $\Gamma$. If $\alpha_1 \dots \alpha_n$ are all the attributes of the start symbol S, this gives functions

$$\text{symbol}(\Gamma,\alpha_1) : \text{DOM}(S) \to V_{\alpha_1} \quad \dots \quad \text{symbol}(\Gamma,\alpha_n) : \text{DOM}(S) \to V_{\alpha_n}$$

for each $\Gamma$ in SYM(S). The product of these functions is a member of CONT(S) and the least upper bound (= join) of these products will be the function s of theorem 1. For each production $X_{p,0} \to X_{p,1}, \dots, X_{p,-1}$ there are a finite number of ways of choosing graphs Q(0) Q(1) ... Q(-1) such that : Q(j) is in SYM($X_{p,j}$) for j = 0, 1, ..., -1;

(∗∗∗) there is an arc from $\alpha$ to $\alpha'$ in Q(0) if and only if there is a directed path from $(X_{p_o,\alpha})$ to $(X_{p_o,\alpha'})$ in $D_p$ [Q(1), ..., Q(-1)].

For each synthesized attribute $\alpha$ in $\underline{X}_{p,0}$ and each such choice of Q = (Q(0), Q(1), ..., Q(-1)) we introduce a function

$$\text{rule}\ (Q,\alpha) : \text{DOM}(X_{p_o}) \to W(Q(0),\alpha) \to V_\alpha .$$

Because the graph $D_p[Q(1), \ldots, Q(-1)$ contains no cycle, it gives a function, rule$(Q, \alpha)$, that does not require recursion or the fix point operator Y. Using these functions we complete the definition of s by

$$\text{symbol}(\Gamma, \alpha) = \text{the join of rule}(Q, \alpha) \text{ such that } Q(0) = \Gamma.$$

We have now proved (a) ; if we can show that every derivation tree in a well defined grammar has a computation sequence, theorem 1 and the lemma will give (b) and (c).

There is one and only one way of assigning symbol graphs and composite graphs to nodes of a derivation tree $\pi$ so that (***) is satisfied – there is a unique choice of symbol graph for each terminal, and, working up the tree, one and only one choice of Q for each application of a production. The composite graphs partially order the attributes at the nodes of the tree, because no composite graph contains a cycle. Let

$$(u_1, \alpha_1) \ldots (u_n, \alpha_n)$$

be an embedding of this partially ordered set in a liniar order. If $\tau$ is a complete assignment to the derivation tree, the value of Next$(\tau)(u_j, \alpha_j)$ is given by rule $(Q, \alpha_j)$ for the Q at node $u_j$ and this only depends on

$$\tau(u_1, \alpha_1) \ldots \tau(u_{j-1}, \alpha_{j-1}) .$$

Our linearly ordered set $(u_1, \alpha_1) \ldots (u_n, \alpha_n)$ is a computation sequence.

<u>Comment</u> In a well defined grammar we have :

(a) every derivation tree has a computation sequence ;

(b) for each derivation tree $\pi$ there is precisely one complete assignment that fits $\pi$. Our lemma shows (a) implies (b) ; the grammar

$$S \to 9 \qquad f(sy_0, c, d) = (c, \underline{\text{if}}\ d \text{ even } \underline{\text{then}}\ 1\ \underline{\text{else}}\ d, c-1)$$

shows (b) does not imply (a) because it is circular but there is precisely one complete assignment that fits its only derivation tree :

$sy_0 = 1$, $c = 1$, $d = 0$. There are grammars satisfying (a) that are not well defined, but they must have useless productions [9].

<u>Example ctd.</u>

The circularity test for our grammar BIN generates

$$\mathrm{SYM}(B) = (\Gamma_0, \Gamma_1) \text{ where } \Gamma_0 = c\ v\ , \qquad \Gamma_1 = c \rightarrow v$$

$$\mathrm{SYM}(L) = (\Gamma_2, \Gamma_3) \text{ where } \Gamma_2 = c\ v\ l\ , \qquad \Gamma_3 = c \rightarrow v\ l\ .$$

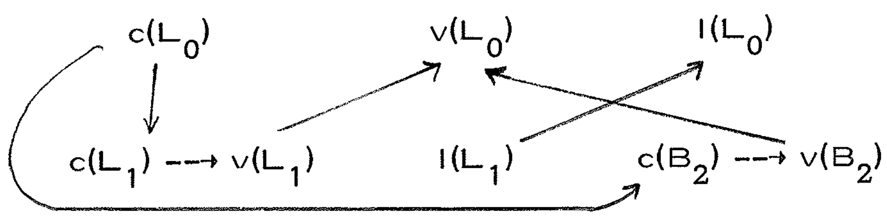
We see that we must introduce functions

$\text{symbol }(\Gamma_0, v) : \mathrm{DOM}(B) \rightarrow V_v$ $\qquad$ $\text{symbol }(\Gamma_1, v) : \mathrm{DOM}(B) \rightarrow V_c \rightarrow V_v$

$\text{symbol }(\Gamma_2, v) : \mathrm{DOM}(L) \rightarrow V_v$ $\qquad$ $\text{symbol }(\Gamma_3, v) : \mathrm{DOM}(L) \rightarrow V_c \rightarrow V_v$

$\text{symbol }(\Gamma_2, l) : \mathrm{DOM}(L) \rightarrow V_l$ $\qquad$ $\text{symbol }(\Gamma_3, l) : \mathrm{DOM}(L) \rightarrow V_l$

The test also generates four graphs in $\mathrm{COMP}(L_0 \rightarrow L_1 B_2)$, the composite graphs given by including or excluding broken arrows in the graph




For the production $L_0 \rightarrow L_1 B_2$ there are four ways of choosing graphs Q(0) Q(1) Q(2) that satisfy (***)

| | $Q_1$ | $Q_2$ | $Q_3$ | $Q_4$ |
|---|---|---|---|---|
| Q(0) | $\Gamma_2$ | $\Gamma_3$ | $\Gamma_3$ | $\Gamma_3$ |
| Q(1) | $\Gamma_2$ | $\Gamma_2$ | $\Gamma_3$ | $\Gamma_3$ |
| Q(2) | $\Gamma_0$ | $\Gamma_1$ | $\Gamma_0$ | $\Gamma_1$ |

Since L has two synthesized attributes, we have eight rule-functions and four symbol functions

$$
\begin{aligned}
\text{rule}\ (Q_1,\ v)[L_1B_2] &= \text{symbol}\ (\Gamma_2,\ v)[L_1] + \text{symbol}\ (\Gamma_0,\ v)[B_2] \\
\text{rule}\ (Q_2,\ v)[L_1B_2]\ c &= \text{symbol}\ (\Gamma_2,\ v)[L_1] + \text{symbol}\ (\Gamma_1,\ v)[B_2]\ c \\
\text{rule}\ (Q_3,\ v)[L_1B_2]\ c &= \text{symbol}\ (\Gamma_3, v)\ [L_1](c+1) + \text{symbol}\ (\Gamma_0, v)[B_2] \\
\text{rule}\ (Q_4,\ v)[L_1B_2]\ c &= \text{symbol}\ (\Gamma_3, v)\ [L_1](c+1) + \text{symbol}\ (\Gamma_1, v)[B_2]\ c
\end{aligned}
$$

$$
\begin{aligned}
\text{symbol}\ (\Gamma_2,\ v)[L_1B_2] &= \text{rule}\ (Q_1,\ v)[L_1B_2] \\
\text{symbol}\ (\Gamma_3,\ v)[L_1B_2]\ c &= \text{rule}\ (Q_2,\ v)[L_1B_2]\ c \cup \text{rule}(Q_3,\ v)[L_1B_2]\ c \\
&\quad \cup \text{rule}\ (Q_4,\ v)[L_1B_2]\ c
\end{aligned}
$$

$$
\begin{aligned}
\text{rule}\ (Q_1,\ l)[L_1B_2] &= \text{symbol}\ (\Gamma_2,\ l)[L_1] + 1 \\
\text{rule}\ (Q_2,\ l)[L_1B_2] &= \text{symbol}\ (\Gamma_2,\ l)[L_1] + 1 \\
\text{rule}\ (Q_3,\ l)[L_1B_2] &= \text{symbol}\ (\Gamma_3,\ l)[L_1] + 1 \\
\text{rule}\ (Q_4,\ l)[L_1B_2] &= \text{symbol}\ (\Gamma_3,\ l)[L_1] + 1
\end{aligned}
$$

$$
\begin{aligned}
\text{symbol}\ (\Gamma_2,\ l)[L_1B_2] &= \text{rule}\ (Q_1,\ l)[L_1B_2] \\
\text{symbol}\ (\Gamma_3,\ l)[L_1B_2] &= \text{rule}\ (Q_2,\ l)[L_1B_2] \cup \text{rule}\ (Q_3,\ l)[L_1B_2] \\
&\quad \cup \text{rule}\ (Q_4,\ l)[L_1B_2]
\end{aligned}
$$

In the next section we show that the above twelve equations can be replaced by:

$$
\begin{aligned}
\text{symbol}\ (\Gamma_3,\ v)[L_1B_2]\ c &= \text{symbol}\ (\Gamma_3,\ v)[L_1]\ (c+1) \\
&\quad + \text{symbol}\ (\Gamma_1,\ v)[B_2]\ (c) \\
\text{symbol}\ (\Gamma_3,\ l)[L_1B_2] &= \text{symbol}\ (\Gamma_3,\ l)[L_1] + 1
\end{aligned}
$$

clearly these are unsugared versions of our original equations:

$$
\begin{aligned}
lv[LB]\ c &= lv[L](c+1) + lv[B]\ c \\
ll\ [LB] &= ll\ [L] + 1.
\end{aligned}
$$

## 4. Other desirable properties

Well defined attribute grammars can have other desirable properties that simplify the task of making a compiler for the language they generate. In this section we introduce six such properties and show how the reformulation within mathematical semantics of an attribute grammar G becomes simpler when G has one of these properties.

### Definition 3

Let $D_p$ be the graph introduced in [9] for a production $p : X_{p0} \rightarrow X_{p1} \ldots X_{p,-1}$ in an attribute grammar. Let $W_p$ be the subgraph of $D_p$ formed by deleting every arrow from an inherited attribute of $X_{p0}$ and every arrow to a synthesized attribute from an attribute of $X_{p,1} \ldots X_{p,-1}$. We say that the production p is:

<u>unordered</u> if $W_p$ is empty;

<u>ordered</u> if each arrow in $W_p$ from an attribute of $X_{p,j}$ to an attribute of $X_{p,k}$ satisfies $0 < j < k$;

<u>reordered</u> if there is a permutation f of $1, 2, \ldots, n(p)-1$ such that each arrow in $W_p$ from an attribute of $X_{p,j}$ to an attribute of $X_{p,k}$ satisfies $j \neq 0 \wedge f(j) < f(k)$;

<u>tangled</u> if there are no cycles in the graph $D_p(ALL(X_{p,1}) \ldots ALL(X_{p,-1}))$ where ALL(X) is the graph with an arrow from every inherited attribute of X to every synthesized attribute of X;

<u>benign</u> if there are no cycles in the graph $D_p(SOME(X_{p,1}) \ldots SOME(X_{p,-1}))$ where SOME(X) is the union of the graphs in SYM(X);

<u>well defined</u> if there are no cycles in any of the graphs $D_p(Q(1), \ldots Q(-1))$ for $Q(1) \in SYM(X_{p,1}) \ldots Q(-1) \in SYM(X_{p,-1})$

## Remark

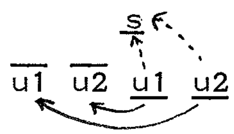
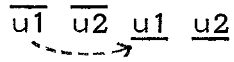
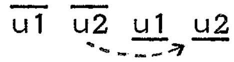
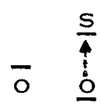
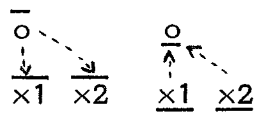
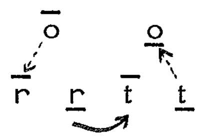
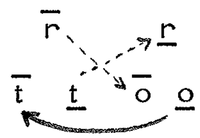
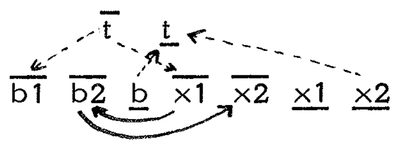
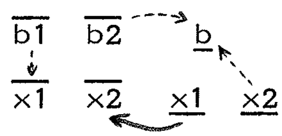
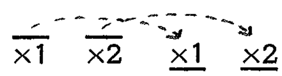
A grammar is not circular if all its productions are well defined. If a production has one of the other properties we have defined, then the order of evaluating the attributes of the symbols on the right side of the production (right symbols) is simplified. For a benign production this order does not depend on the productions used to expand the right symbols. For a tangled production this order can be such that all the inherited attributes of a right symbol occur before any of its synthesized attributes. For an ordered (reordered, unordered) production this order can be such that one can evaluate all attributes of a right symbol $X_{p,i}$ before evaluating any attribute of the next right symbol (the symbol following $X_{p,i}$ in some permutation of the right side of the production, any other right symbol). Clearly these distinctions are significant when designing a compiler for the language given by an attribute grammar.

## New example

Consider the attribute grammar CONTRIVED

$\overline{S} = \{ \}$ $\overline{U} = \{\overline{u}1, \overline{u}2\}$ $\overline{O} = \{\overline{o}\}$ $\overline{R} = \{\overline{r}\}$ $\overline{T} = \{\overline{t}\}$ $\overline{B} = \{\overline{b}1, \overline{b}2\}$ $\overline{X} = \{\overline{x}1, \overline{x}2\}$

$\underline{S} = \{\underline{s}\}$ $\underline{U} = \{\underline{u}1, \underline{u}2\}$ $\underline{O} = \{\underline{o}\}$ $\underline{R} = \{\underline{r}\}$ $\underline{T} = \{\underline{t}\}$ $\underline{B} = \{\underline{b}\}$ $\underline{X} = \{\underline{x}1, \underline{x}2\}$

| name | production | semantic rule | production graph |
|---|---|---|---|
| a | S → U | $f_a(\underline{u1}, \underline{u2})(\underline{s}, \overline{u1}, \overline{u2}) = (\underline{u1}+\underline{u2}, \underline{u2}, \underline{u1})$ |  |
| b | U → 5 | $f_b(\overline{u1}, \overline{u2})(\underline{u1}, \underline{u2}) = (23\times\overline{u1}, 5)$ |  |
| c | U → 7 | $f_c(\overline{u1}, \overline{u2})(\underline{u1}, \underline{u2}) = (7, 29\times\overline{u2})$ |  |
| d | S → O | $f_d(\underline{o})(\underline{s}, \overline{o}) = (3\times\underline{o}, 2)$ |  |
| e | O → X | $f_e(\overline{o}, \underline{x1}, \underline{x2})(\underline{o}, \overline{x1}, \overline{x2}) = (\underline{x1}/\underline{x2}, \overline{o}, \overline{o})$ |  |
| f | O → RT | $f_f(\overline{o}, \underline{r}, \underline{t})(\underline{o}, \overline{r}, \overline{t}) = (19\times\underline{t}, 31\times\overline{o}, 37\times\underline{r})$ |  |
| g | R → TO | $f_g(\overline{r}, \underline{t}, \underline{o})(\underline{r}, \overline{t}, \overline{o}) = (11\times\underline{t}, 13\times\underline{o}, 17\times\overline{r})$ |  |
| h | T → BX | $f_h(\overline{t}, \underline{b}, \underline{x1}, \underline{x2})(\underline{t}, \overline{b1}, \overline{b2}, \overline{x1}, \overline{x2}) = (\underline{b}+\underline{x2}, \overline{t}, \overline{x1}, \overline{t}, \overline{b2})$ |  |
| k | B → X | $f_k(\overline{b1}, \overline{b2}, \underline{x1}, \underline{x2})(\underline{b}, \overline{x1}, \overline{x2}) = (\overline{b2}-\underline{x2}, \overline{b1}, \underline{x1})$ |  |
| l | X → 9 | $f_l(\overline{x1}, \overline{x2})(\underline{x1}, \underline{x2}) = (\overline{x1}, \overline{x2})$ |  |

The broken arrows in a production graph $D_p$ are those that are not in $W_p$. We see that the productions $U \to 5$, $U \to 7$, $S \to O$, $O \to X\ X$ $\to 9$ are unordered, the production $O \to RT$ is ordered, and the production $R \to TO$ is reordered.

<u>Theorem 3</u>

(a) unordered $\to$ ordered $\to$ reordered $\to$ tangled $\to$ benign $\to$ well defined.

(b) the chain of implications in (a) is proper.

(c) a production $X_{p,0} \to X_{p,1} \ldots X_{p,-1}$ is tangled if and only if the attributes of $X_{p,0}, X_{p,1} \ldots X_{p,-1}$ can be ordered in such a way that every inherited attribute of $X_{p,i}$ can be evaluated before a synthesized attribute of $X_{p,i}$ is evaluated.

<u>Proof</u>

(a) The first, second, fourth and fifth implications follow directly from the definitions. For the third implication assume that production $X_{p,0} \to X_{p,1} \ldots X_{p,-1}$ is reordered and $D_p(ALL(X_{p,1}) \ldots ALL(X_{p,-1}))$ has a cycle. This cycle cannot pass through an inherited attribute of $X_{p,0}$ because there are no arrows <u>to</u> these attributes; it cannot pass through a synthesized attribute of $X_{p,0}$ because there are no arrows <u>from</u> these attributes in a reordered production; it cannot use an arrow of $W_p$ because f increases along such an arrow and f is constant on the arrows of ALL(X). Since ALL(X) has no cycle, our assumption is absurd.

(b) Consider the grammar CONTRIVED. The production $O \to RT$ is ordered, but not unordered; the production $R \to TO$ is reordered, but not ordered; the production $T \to BX$ is tangled, but not reordered; the production $B \to X$ is not tangled.

As the production graph $D_l$ is the only symbol graph in SYM(X) it is also the graph SOME(X). Since $D_k[D_l]$ has no cycles, the production $B \to X$ is benign. Now consider the attribute grammar given by the first three productions of CONTRIVED. The production $S \to U$ is well defined because there are no cycles in $D_a[D_b]$ and $D_a[D_c]$, but it is not benign because there is a cycle in $D_a[D_b \cup D_c]$.

(c→) If the composite graph $D_p(ALL(X_{p,1}) \ldots ALL(X_{p,-1}))$ has no cycles, it is the graph of a partial order on the attributes of $X_{p,0}, X_{p,1} \ldots X_{p,-1}$. Because any partial order can be embedded in a linear order we can evaluate attributes in an order satisfying:

(1) if attribute $\beta$ depends on attribute $\alpha$.

(2) if $\alpha$ is an inherited attribute of $X_{p,j}$ and $\beta$ is a synthesized attribute of $X_{p,j}$, then $\alpha$ is evaluated before $\beta$.

(c←) Assume the attributes of $X_{p,0}, X_{p,1} \ldots X_{p,-1}$ can be evaluated in some order satisfying (1) and (2). Consider an edge from attribute $\alpha$ to attribute $\beta$ in $D_p(ALL(X_{p,1}) \ldots ALL(X_{p,-1}))$. If the edge is in $D_p$, $\alpha$ is evaluated before $\beta$ by (1); if the edge is in $ALL(X_{p,j})$, $\alpha$ is evaluated before $\beta$ by (2). Since "is evaluated before" is a linear order, $D_p(ALL(X_{p,1}) \ldots ALL(X_{p,-1}))$ has no cycles.

<u>Comment</u> An attribute grammar with only ordered productions allows "evaluation in one pass from left to right" [3, 8]. One applies the following recursive algorithm for each application of a production $X_{p,0} \to X_{p,1} \ldots X_{p,-1}$ in a derivation tree:

<u>lr evaluate</u> : <u>begin</u> fetch inherited attributes of $X_{p,0}$;
    <u>for</u> $X := X_{p,1}$ <u>to</u> $X_{p,-1}$
    <u>do begin</u> Use $f_p$ to evaluate inherited attributes of $X$;
        Call <u>lr evaluate</u> to calculate synthesized attributes of $X$;
    <u>end</u>
    Use $f_p$ to calculate synthesized attributes of $X_{p,0}$;
<u>end</u>;

A similar argument shows that one can evaluate an attribute grammar with only tangled productions in one pass, if we allow a pre-evaluation phase in which we either rearrange the derivation tree or add a next sibling pointer to each node in the tree. One applies the following recursive algorithm for each application of a production $X_{p,0} \to X_{p,1} \dots X_{p,-1}$ in the tree

<u>t-evaluate</u> : <u>begin</u> fetch inherited attributes of $X_{p,0}$;
<u>for</u> $\alpha$ := attribute of $X_{p,0}\, X_{p,1} \dots X_{p,-1}$
in order given by theorem 3 c
<u>do</u> <u>if</u> $\alpha$ is inherited attribute of $X_{p,j}$ for $j \neq 0$
<u>or</u> $\alpha$ is synthesized attribute of $X_{p,0}$
<u>then</u> Use $f_p$ to calculate $\alpha$
<u>else</u> <u>if</u> $\alpha$ not already calculated
<u>then</u> Call <u>t-evaluate</u> to calculate
<u>all</u> synthesized attributes of the
$X_{p,j}$ to which $\alpha$ belongs
<u>end</u>.

An algorithm for finding the finite number of passes required to evaluate a well defined attribute grammar is given in [15].

Definition 4

An attribute grammar is in normal form if for every production the function

$$f_p : L_p^o \to R_p^o \to R_p^o$$

satisfies

$$f_p(l)(r) = f_p(l)(r')$$

for any $l$ in $L_p^o$ and any $r, r'$ in $R_p^o$.

Comment For well defined attribute grammars in normal form, many tiresome distinctions disappear.

Theorem 4 (Hanne Riis) If an attribute grammar is in normal form then the production p is reordered $\Leftrightarrow$ production p is tangled.

Proof

Suppose $X_{p,0} \to X_{p,1} \ldots X_{p,-1}$ is a tangled production of an attribute grammar G. If G is in normal form, we can evaluate all attributes of a right symbol $X_{p,i}$ "at the same time", because we can wait until a synthesized attribute is required before evaluating the inherited attributes. To make this argument precise we introduce the relation R by:

$j\,R\,k \Leftrightarrow$ there is an arrow in $D_p$ from a synthesized attribute of $X_{p,j}$ to an inherited attribute of $X_{p,k}$

The reflexive transitive closure $R^*$ of this relation is a partial order because $jR^*k$, $kR^*j$, $j \neq k$ implies a chain of arrows in $D_p$ that becomes a cycle in $D_p[\mathrm{ALL}(X_{p,1}) \ldots \mathrm{ALL}(X_{p,-1})]$; and this cannot happen when p is a tangled production.
Embed the partial order $R^*$ in a total order and define $f(j)$ as the position of $X_{p,j}$ in this total order. Clearly f is a permutation and $j\,R\,k$ implies $f(j) < f(k)$. Because G is in normal form there are no arrows in $W_p$ from either synthesized attributes of $X_{p,0}$ or inherited attributes of $X_{p,1} \ldots X_{p,-1}$. If there is an

arrow in $W_p$ from an attribute of $X_{p,j}$ to an attribute of $X_{p,k}$ we must have $j \neq 0$ and $j R k$. The tangled production p must be reordered ; the converse implication is given by theorem 3.

New example ctd.

As an illustration of the simplifications possible when productions have our "compiler friendly" properties we reformulate our grammar CONTRIVED within mathematical semantics.

| Syntactic Rule | Semantic function |
|---|---|
| $S \to U$ | $s[U] = \underline{u1} + \underline{u2}\ \underline{\text{where}}\ (\underline{u1}, \underline{u2}) = u[U](\underline{u2}, \underline{u1})$ |
| $U \to 5$ | $u[5](\overline{u1}, \overline{u2}) = (23 \times \overline{u1},\ 5)$ |
| $U \to 7$ | $u[7](\overline{u1}, \overline{u2}) = (7,\ 29 \times \overline{u2})$ |
| $S \to O$ | $s[O] = 3 \times o[O]2$ |
| $O \to X$ | $o[X](\overline{o}) = \underline{x1} / \underline{x2}\ \underline{\text{where}}\ (\underline{x1}, \underline{x2}) = x[X](\overline{o}, \overline{o})$ |
| $O \to RT$ | $o[RT](\overline{o}) = 19 \times \underline{t}\ \underline{\text{where}}\ \underline{r} = r[R](31 \times \overline{o})$ $\underline{\text{and}}\ \underline{t} = t[T](37 \times \underline{r})$ |
| $R \to TO$ | $r[TO](\overline{r}) = 11 \times \underline{t}\ \underline{\text{where}}\ \underline{o} = o[O](17 \times \overline{r})$ $\underline{\text{and}}\ \underline{t} = t[T](13 \times \underline{o})$ |
| $T \to BX$ | $t[BX](\overline{t}) = \underline{b} + \underline{x2}\ \underline{\text{where}}\ \underline{b} = b[B](\overline{t}, \overline{t})$ $\underline{\text{and}}\ (\underline{x1}, \underline{x2}) = x[X](\overline{t}, \overline{t})$ |
| $B \to X$ | $b[X](\overline{b1}, \overline{b2}) = \overline{b2} - \underline{x2}\ \underline{\text{where}}\ (\underline{x1}, \underline{x2}) = x[X](\overline{b1}, \underline{x1})$ |
| $X \to 9$ | $x[X](\overline{x1}, \overline{x2}) = (\overline{x1}, \overline{x2})$ |

The only productions which are not tangled are $S \to U$ and $B \to X$. For these two productions and no others we have recursion in the correspond semantic function. Since our grammar is well defined, this recursion can be eliminated by theorem 2. For the non-benign production $S \to U$, the proof of the theorem suggests replacing the semantic functions for the first three productions by

$$s[U] = sb[U] \cup sc[U]$$

$$sb[U] = \underline{u1} + \underline{u2}\ \underline{\text{where}}\ \underline{u2} = u2\,b[U]$$
$$\underline{\text{and}}\quad \underline{u1} = u1\,b[U](\underline{u2})$$

$$sc[U] = \underline{u1} + \underline{u2}\ \underline{\text{where}}\ \underline{u1} = u1\,c[U]$$
$$\underline{\text{and}}\quad \underline{u2} = u2\,c[U](\underline{u1})$$

$$u1\,b[5](\overline{u1}) = 23 \times \overline{u1} \qquad u2\,b[5] = 5$$

$$u1\,c[7] = 7 \qquad u2\,c[7](\overline{u2}) = 29 \times \overline{u2}$$

The proof of our next theorem shows why the join operator $\cup$ is not needed when removing the recursion in the semantic function for the benign production $B \to X$:

$$b[X](\overline{b1}, \overline{b2}) = \overline{b2} - \underline{x2}\quad \underline{\text{where}}\ \underline{x1} = \overline{b1}$$
$$\underline{\text{and}}\quad \underline{x2} = \underline{x1}$$

Convention We use MS[G] as an abbreviation for : a specification in mathematical semantics of the function s in the proof of theorem 1 for an attribute grammar G.

Determinacy Theorem If all productions in an attribute grammar G are benign, then the join operator $\cup$ need not appear in any of the functions specified by MS[G].

Proof

For each X the set of symbol graphs SYM(X) can be replaced by their union SOME(X). If we make this replacement in the proof of theorem 2 there is one and only one Q satisfying requirement (***) for a production $X_{p,0} \to X_{p,1} \dots X_{p,-1}$.
The functions rule $(Q, \alpha)$ that are joined in the definition of symbol $(\Gamma, \alpha)$ come from different productions with the same left side. Such joins do not appear in an MS[G] specification because of the convention in section 2.

Comment  Because our grammar MS(G) works on derivation trees, the implicit joins in the section 2 convention do not destroy determinacy. The convention that MS(G) semantic functions may be specified in terms of one another seems just as harmless. In our statement of theorem 2 we avoided the fix point operator Y used to unravel this mutual recursion.

Splitting Theorem  If all productions in an attribute grammar G are tangled, then we can construct an MS(G) such that

(a) no function specified in MS(G) uses the operators Y and $\cup$

(b) every function specified in MS(G) is in CONT(X) for some X in $N \cup T$.

Proof

(a) : Combine theorem 2 and the Determinacy theorem.

(b) : Consider the MS(G) formulation given by part (a). It consists of specifications of the functions Symbol (SOME(X), $\alpha$) for each X in $N \cup T$ and each synthesized attribute $\alpha$ in $\underline{X}$. For a tangled production $X_{p,0} \to X_{p,1} \dots X_{p,-1}$, all inherited attributes of $X_{p,i}$ can be evaluated before any synthesized attributes of $X_{p,i}$. Thus each function Symbol (SOME(X), $\alpha$) can be extended from DOM(X) $\to$ W(SOME(X), $\alpha$) $\to V_\alpha$ to DOM(X) $\to$ INH(X) $\to V_\alpha$. Our theorem now follows from the fact that the lattice product of DOM(X) $\to$ INH(X) $\to V_\alpha$ for $\alpha$ in $\underline{X}$ is isomorphic to the lattice CONT(X) = DOM(X) $\to$ INH(X) $\to$ SYN(X).

Comment  When we removed recursion from the semantic function for the benign production $B \to X$ in our grammar, the required splitting of SYN(X) was implicit. The general construction would give

$$b[X](\overline{b1},\overline{b2}) \;=\; \overline{b2} - \underline{x2} \;\; \underline{\text{where } x1} = x1[X]\overline{b1}$$
$$\underline{\text{and}} \;\; \underline{x2} = x2[X]\underline{x1}$$

$$x1[9](\overline{x1}) \;=\; \overline{x1}$$
$$x2[9](\overline{x2}) \;=\; \overline{x2}$$

and minor changes in the specifications for productions $O \to X$ and $T \to BX$.

## Concluding remarks

The converse of the problem in this paper - forming an attribute grammar from a specification in mathematical semantics - is the subject of [8, 11]. Is there any good reason for basing a compiler generator on attribute grammars, rather than mathematical semantics [14] ? If there is, should one allow for attribute grammars that are well defined but not benign ? Any algorithm for checking that an attribute grammar is well defined is computationally intractable [6, 7]. Chircia and Martin [4] give a pragmatic reason for preferring benign grammars for particular languages; our determinacy theorem gives a theoretical reason for this preferrence.
The author would like to thank Ole Lehrmann Madsen, Hanne Riis, and Erik Meineche Schmidt for many fruitful discussions on this and the other topics discussed in this paper.

## References

[1] L. Aiello, M. Aiello, R.W. Weyrauch, The semantics of PASCAL in LCF, STAN-74-447, Stanford University 1974.

[2] D. Bjørner, C.B. Jones, Vienna Development Method : The meta language, Springer Lecture Notes, 1978.

[3] L.M. Chirica, D.F. Martin, An order-algebraic definition of Knuthian semantics, Math. Sys. Th 13(1979) 1-27.

[4] L.M. Chirica, D.F. Martin, An algebraic formulation of Knuthian semantics, Symposium Found. Comp. Sci. 17(1976), 127-136.

[5] J.B. Dennis, On storage management for advanced programming languages, Project MAC Computation Structures Group, memo 109-1, MIT 1974.

[6] M. Jazayeri, W.F. Ogden, W.C. Rounds, The intrinsically exponential complexity of the circularity problem for attribute grammars, Comm. ACM 12 (1975) 697-721.

[7] N. Jones, Circularity testing of attribute grammars requires exponential time : a simpler proof, DAIMI PB-107, Aarhus 1980.

[8] H. Ganzinger, Some principles for the development of compiler descriptions from denotational language definitions, Tech. Univ. München, Preprint, 1980.

[9] D.E. Knuth, Semantics of Context free languages, Math. Sys. Theory 2 (1968) 127-145; correction, ibid 5 (1971) 95.

[10] P.M. Lewis, P.J. Rosenkrantz, R.E. Stearns, Attributed translations, J. Comp. Sys. Sci. 9 (1974) 279-307.

[11] O.L. Madsen, On defining semantics by means of extended attribute grammars, DAIMI PB-90, Aarhus 1978.

[12] R.E. Milne, The formal semantics of computer language and their implementations, Ph.D. thesis, Cambridge University 1974.

[13] P.D. Mosses, The mathematical semantics of Algol 60, Program Research Group PRG-12, Oxford 1974.

[14] P.D. Mosses, Mathematical Semantics and Compiler Generation, Ph.D. thesis, Oxford University 1975.

[15] H. Riis, S. Skyum, K-visit attribute grammars, DAIMI PB-121, Aarhus 1980.

[16] D. Scott, Mathematical concepts in programming language semantics, AFIPS proceedings 40 (STCC 1972) 225-242.

[17] D. Scott, C. Strachey, Towards a Mathematical Semantics for Computer Languages, Proc. Symp. Computers and Automata, Brooklyn Polytechnic 1971.

[18] R.D. Tennent, Mathematical Semantics and the design of programming languages, Ph.D. thesis, Toronto University 1973.

[19] R.D. Tennent, The denotational semantics of programming languages, Comm. ACM 8 (1976) 437-453.