A Translation from Attribute Grammars to Catamorphisms

Maarten Fokkinga, Johan Jeuring, Lambert Meertens, Erik Meijer

Version of November 9, 1990 — reformatted at March 10, 1994

Let AG be an attribute grammar, with underlying context free grammar G and attribute evaluation rules A. The function that decorates —according to A— a parse tree with attribute values and then delivers the synthesized attribute value of the root node, is denoted $[\![A]\!]$. We translate G into a functor F such that any parse tree for G is an element of the initial F-algebra. The attribute evaluation rules A are translated to a function φ such that $(\!|\mathsf{F}|\ \varphi|\!)$ is, in a precise sense, equivalent to $[\![A]\!]$.

1 The translation

We begin by fixing some terminology and notations. Let AG be an attribute grammar. We define

G	=	the underlying context free grammar of AG.
X	=	the type of the inherited attributes (explained in (2, 3)).
Y	=	the type of the synthesized attributes (explained in (2, 3)).
T	=	the set of parse trees for grammar G (explained in (4)).
A	=	the attribute evaluation rules of AG (explained in (5)).
$[\![A]\!]$	=	the function that, given $t \in T$ and $x \in X$, $\{$ decorates tree t according to rules A when the inherited attribute of the root node of t is set to x, and $\}$ yields as result the synthesized attribute value $\in Y$ of the root node of t; thus $[\![A]\!] : T \times X \to Y$. (Explained in (6).)

In this note we construct a catamorphism for AG that is equivalent (equal) to $[\![A]\!]$. (Neither $[\![A]\!]$ nor the catamorphism is intended to yield a parse tree when given an actual string. However, one can extend any AG to AG' in such a way that $[\![A']\!](t, x)$ yields the fully decorated parse tree; see Example 2.)

Plan We shall proceed as follows.

- First we show that we may assume that AG has a simple form, so that the actual translation can be formulated without too many indices and the like.

- The context free grammar G determines a functor F.
- **Lemma** T is a subset of the carrier of the initial F-algebra.
- The attribute evaluation rules A determine a function $\varphi : (X \to Y)\mathsf{F} \to (X \to Y)$.
- **Theorem** $[\![A]\!](t, x) = (\!|\mathsf{F}|\ \varphi|\!)\, t\, x$.

We assume that we are working in the category **Set**, or in a **Set**-like category, like CPO.

Simplification of the attribute grammar For notational simplicity we make the following three assumptions, without loss of generality.

(1) Any terminal a is produced only by rules of the form $lhs \to a$ where a in the right hand side has no attributes. This can be achieved by the addition of auxiliary nonterminal symbols, say one for each terminal symbol a.

(2) Any nonterminal has precisely one inherited and one synthesized attribute. This can be achieved for any AG by tupling the inherited attributes of each nonterminal, and also the synthesized ones. Notice that a tuple may be the empty tuple $() \in \mathbf{1}$.

(3) All inherited attributes have one type X say, and all synthesized attributed have one type Y say. This can be achieved thanks to the following technique. Suppose we have types $A_0, \ldots, A_{m-1}, B_0, \ldots, B_{n-1}$ and f is a function of type $A_i \to B_j$. Then we can define $f' : A_0 + \cdots + A_{m-1} + \mathbf{1} \to B_0 + \cdots + B_{n-1} + \mathbf{1}$ by

$$\begin{array}{lcll} f' & = & f_0 \triangledown \cdots \triangledown f_n & \\ f_{i'} & = & \mathsf{inj}_n \circ !A_{i'} & \text{for } i' \neq i \\ & = & \mathsf{inj}_j \circ f & \text{for } i' = i \\ !A & = & \text{the unique fct } A \to \mathbf{1} & \text{for any } A \quad . \end{array}$$

Function f' is equivalent to f in the following sense: $f' \circ \mathsf{inj}_i = \mathsf{inj}_j \circ f$. So, if the actual attribute types are $X_0, \ldots, X_{m-1}$ and $Y_0, \ldots, Y_{n-1}$, then we can take $X = X_0 + \cdots + X_{m-1} + \mathbf{1}$ and $Y = Y_0 + \cdots Y_{n-1} + \mathbf{1}$, and adapt AG accordingly to AG' with $G' = G$ and $[\![A']\!]\hat{}t \circ \mathsf{inj}_0 = \mathsf{inj}_0 \circ [\![A]\!]\hat{}t$ where $\hat{}$ denotes currying and X_0 and Y_0 are the attribute types of the root node.

Construction of the functor Consider an arbitrary production rule of G, say

$$p : \qquad u \to v_0 \ldots v_{n-1} \quad .$$

Here p is just a (unique) label of the rule, and to be *very* precise we should have subscripted all the entries in the rule with p. Let P be the set of production rule labels of G; in the sequel we let P be the domain over which p ranges. Rule p determines a functor F_p, and all rules together determine a functor F and an F-algebra, as follows.

$$\mathsf{F}_p \quad = \quad \mathbf{1} \qquad \text{if } n = 1 \text{ and } v_0 \text{ is terminal; recall (1)}$$

$$
\begin{array}{lcll}
 & = & \mathrm{I} \times \ldots \times \mathrm{I} \quad (n \text{ times}) & \text{otherwise. The product is } \mathbf{1} \text{ if } n = 0 . \\
\mathsf{F} & = & \sum p :: \mathsf{F}_p & \\
(T', in) & = & \text{the/an initial } \mathsf{F}\text{-algebra} & \\
in_p & = & in \circ \mathsf{inj}_p \quad : T'\mathsf{F}_p \to T' & \text{for all } p \\
in & = & \nabla p :: in_p, & \text{follows from preceding lines} \quad .
\end{array}
$$

(I is the identity functor, and $\mathbf{1}$ is the constant functor. Mono-functor $\mathsf{F} \times \mathsf{G}$ is defined by $x(\mathsf{F} \times \mathsf{G}) = x\mathsf{F} \times x\mathsf{G}$ for any type and function x. Similarly, $x(\sum p :: \mathsf{F}_p) = \sum p :: x\mathsf{F}_p$ for any type and function x.)

(4) Lemma *There exists an embedding from the parse trees of G into (T', in).*

Proof As we have not yet given a definition of (the algebra of) parse trees, we do it here. A *parse tree t for production rule p* consists of an indication "$node_p$" and n (possibly 0) immediate constituents $t_0, \ldots, t_{n-1}$ such that, for all i, t_i is a parse tree for a production rule that has v_i in its left hand side. Let us use the notation "$node_p(t_0, \ldots, t_{n-1})$" for t. Thus "$node_p$" is made into a partial operation of type $T^n \to T$. It is partial since the arguments of operation $node_p$ have to satisfy a condition.

Now define function $\epsilon : (T,\ \nabla p :: node_p) \to (T',\ \nabla p :: in_p)$ by $\epsilon(node_p(t_0, \ldots, t_{n-1})) = in_p(\epsilon t_0, \ldots, \epsilon t_{n-1})$. Thus ϵ is an F-homomorphism from T to T' that has a post-inverse; it is an embedding. (Note that ϵ does not necessarily have a pre-inverse, since not every $in_p(t_0, \ldots, t_{n-1}) \in T'$ satisfies necessarily the condition that each t_i has v_i in its root. This happens in Example 2.)

In the sequel we identify $node_p$ with in_p. □

Attribute evaluation Consider again production rule p, now provided with the attribute evaluation rules:

$$
p : \qquad u \to v_0 \cdots v_{n-1} \qquad \text{with } (f, g_0, \ldots, g_{n-1}) \quad ,
$$

or slightly more suggestive (the λ is explained below)

$$
(5) \qquad p : \qquad u(\lambda x,\ f(x,y)) \quad \to \quad \cdots\ v_i(g_i(x,y),\ \lambda y_i)\ \cdots
$$

where y abbreviates $(y_0, \ldots, y_{n-1})$, an abbreviation that is valid throughout the sequel. The occurrences 'λx' and 'λy_i' are binding occurrences, their scope is the entire rule and systematic renaming is allowed. This rule says, for a tree $t = in_p(t_0, \ldots, t_{n-1})$ in which every node has been assigned two values (called *attributes*), that

$$
\begin{array}{lcll}
\text{the second attribute of } t & = & f(x,y) & \\
\text{the first attribute of } t_i & = & g_i(x,y) & \text{for all } i
\end{array}
$$

where x is the first attribute of t and y_i is the second attribute of t_i. Now $[\![A]\!]$ is defined to be a/the function such that

(6) $[\![A]\!](t,x)$ = Let t' be tree t in which every node has been assigned two values (called attributes) in such a way that the first attribute of the root of t' equals x and all subtrees satisfy the above condition (for the appropriate p). Then yield as result the second attribute of the root of t'.

We assume that AG is such that a $[\![A]\!]$ exists; in the proof of Theorem (8) we shall further narrow the choice for $[\![A]\!]$. The specification implies

(7) $[\![A]\!](in_p(t_0,\ldots,t_{n-1}),x) = f(x,y)$ where $i :: y_i = [\![A]\!](t_i, g_i(x,y))$.

The equation suggests to compute the second attribute of any (sub)tree by "attribute evaluation" within that (sub)tree; hence this attribute is called *synthesized*. Similarly, the equation suggests that the first attribute of any (sub)tree is to be determined by the context, i.e., by attribute evaluation in the enclosing tree or by the environment in case the (sub)tree is the entire parse tree; hence this attribute is called *inherited*. As argued in assumptions (2, 3), the typing within rule p (5) is: $x : X$, $y : Y^n$, $f : X \times Y^n \to Y$, and $g_i : X \times Y^n \to X$.

Construction of the catamorphism Attribute grammar rule p (5) determines a function φ_p, and all rules together determine a function φ and a catamorphism $(\!|\varphi|\!)$ as follows.

$$\begin{array}{lcl}
\varphi_p(\varphi_0,\ldots,\varphi_{n-1}) & = & (\lambda x :: f(x,y) \text{ where } i :: y_i = \varphi_i(g_i(x,y))) \quad \text{possibly } n=0 \\
\varphi_p & : & (X \to Y)^n \to X \to Y \\
\varphi & = & (\nabla p :: \varphi_p) \\
 & : & (X \to Y)^{\mathsf{F}} \to (X \to Y) \\
(\!|\mathsf{F}|\ \varphi|\!) & : & T' \to (X \to Y) \quad .
\end{array}$$

Notice that the where-clause defines y by recursion; this corresponds to the potential circularity in the attribute evaluation when $[\![A]\!](t,x)$ is computed as suggested by grammar rule p (5).

(8) Theorem $[\![A]\!](t,x) = (\!|\mathsf{F}|\ \varphi|\!)\, t\, x$.

The equality holds for all $t \in T'$, not only for $t \in T$. This is possible thanks to our assumption (2) that any node in a parse tree has precisely two attributes; these have not been named, and have been referred to as "the first" and "the second" attribute of the node, in the attribute evaluation rule (6). Also, we have assumed in (3) that all functions f and g_i accept all kinds of values (though they may return a result in the summand $\mathbf{1}$ for wrong inputs). The theorem could have been formulated as $[\![A]\!]^\wedge = (\!|\varphi|\!)$, where $^\wedge$ denotes the currying operation.

Proof By induction on the structure of t. Suppose $t = in_p(t_0, \ldots, t_{n-1})$. (Notice that possibly $n = 0$ so that $in_p : \mathbf{1} \to (X \to Y)$; this covers the so-called base case.) We calculate

$$
\begin{array}{ll}
 & (\!|\varphi|\!)\, t\, x \\
= & \quad \text{case assumption on } t \\
 & (\!|\varphi|\!)\, (in_p(t_0, \ldots, t_{n-1}))\, x \\
= & \quad \text{evaluation rule for catamorphisms} \\
 & \varphi_p\, ((\!|\varphi|\!) t_0, \ldots, (\!|\varphi|\!) t_{n-1})\, x \\
= & \quad \text{unfold definition of } \varphi_p \\
 & f(x,y) \text{ where } i :: y_i = (\!|\varphi|\!)\, t_i\, (g_i(x,y)) \\
(*) \quad = & \quad \text{induction hypothesis} \\
 & f(x,y) \text{ where } i :: y_i = [\![A]\!](t_i, g_i(x,y)) \\
= & \quad \text{attribute evaluation equation (7)} \\
 & [\![A]\!](in_p(t_0, \ldots, t_{n-1}),\ x) \\
= & \quad \text{case assumption on } t \\
 & [\![A]\!](t, x) \quad .
\end{array}
$$

In step $(*)$ of the calculation it turns out that the circularity in the attribute evaluation and the mutual recursion in the definition of φ_p should be resolved in the same way. For example, if in the definition of φ_p the y_i are defined to be the *least* fixed points of the equations $i :: y_i = \varphi_i(g_i(x,y))$, then so must specification (6) of $[\![A]\!]$ be understood. $\Box$

2 Examples

Linear parse trees Consider the following attribute grammar AG:

$$
\begin{array}{ll}
p: & \quad u(\lambda x, fy) \to u(gx, \lambda y). \\
q: & \quad u(\lambda x, hx) \to .
\end{array}
$$

This is a very simple example since there is no circularity at all. The parse trees are linear. Attribute evaluation of a tree t of depth n gives $(f^n \circ h \circ g^n)x$ as synthesized attribute value of the root node when its inherited attribute value is set to x. In other words, $[\![A]\!](t,x) = (f^n \circ h \circ g^n)x$. Our construction of the previous section gives

$$
\begin{array}{lcll}
\mathsf{F} & = & \mathrm{I} + \mathbf{1} & \\
T' & \cong & \mathbb{N} & \\
\varphi_p(\varphi) & = & (\lambda x :: fy \text{ where } y = \varphi(g(x,y))) & \text{so } \varphi_p = (g \circ\!\!\to f) \\
\varphi_q() & = & (\lambda x :: hx) & \text{so } \varphi_q = h^\bullet \\
\varphi & = & \varphi_p \triangledown \varphi_q & \text{so } \varphi = (g \circ\!\!\to f) \triangledown h^\bullet
\end{array}
$$

$$
\begin{array}{lcll}
(\!|\mathsf{F}|\ \varphi|\!) & : & \mathbb{N} \to (X \to Y) & \\
(\!|\varphi|\!)t & = & (g \circ\!\!\to f) \circ \cdots \circ (g \circ\!\!\to f) \circ h & = f^n \circ h \circ g^n
\end{array}
$$

where, in the last line, t is assumed to have depth n, i.e., $t = (in_p \circ \cdots \circ in_p \circ in_q)()$.

Binary parse trees The following attribute grammar works on binary (parse) trees with numbers at the tips. The attribute evaluation yields (as synthesized attribute value) a tree of the same shape as the input parse tree t, having all tip numbers equal to the minimum tip value in t. This function has been discussed by a number of people, e.g., Bird [1], Kuiper and Swierstra [3, 4], Fokkinga [2]. We use $+\!\!+$ as join-operation of trees, [.] as tip-former, min as minimum-operation, and we let s, t vary over trees and k, m, n over numbers. The type of trees with numbers at the tips is denoted $\mathbb{N}*$. Let us first present the attribute grammar in the conventional form, i.e., not yet simplified.

$$
\begin{array}{ll}
p: & u(t) \to v(m, \lambda m, \lambda t). \\
q: & v(\lambda k, m \min n, s +\!\!+ t) \to v(k, \lambda m, \lambda s)\ v(k, \lambda n, \lambda t). \\
r_i: & v(\lambda k, i, [k]) \to i. \qquad \text{for all numbers } i
\end{array}
$$

In rule p we see that the (synthesized) first attribute value of v is specified to be equal to the (inherited) second attribute value, and in rules q and r we see (by induction) that the second attribute value of v is specified to be the minimum of the input parse tree. So, eventually, in the third attribute value of v the required tree is delivered. (Kuiper and Swierstra [3] need *ten* lines for this grammar.)

Nonterminal u has no inherited attribute; it may considered to have a "nullary" attribute of type $\mathbf{1}$. Rather than taking $X = \mathbf{1} + \mathbb{N}$ for all inherited attributes, we give u a *dummy* inherited attribute of type $\mathbb{N}$; this avoids the introduction of many injections and inspections. Also, we give u an extra synthesized attribute, and then tuple the two synthesized attributes everywhere giving values of type $Y = \mathbb{N} \times \mathbb{N}*$; in order to avoid many projections we use "parameter matching" at the binding lambdas. Thus we get the following attribute grammar that satisfies the assumptions (1, 2, 3):

$$
\begin{array}{ll}
p: & u(\lambda k, (m, t)) \to v(m, \lambda(m, t)). \\
q: & v(\lambda k, (m \min n,\ s +\!\!+ t)) \to v(k, \lambda(m, s))\ v(k, \lambda(n, t)). \\
r_i: & v(\lambda k, (i, [k])) \to i. \qquad \text{for all } i
\end{array}
$$

The construction of the catamorphism gives now the following.

$$
\begin{array}{lcll}
\mathsf{F} & = & \mathrm{I} + \mathrm{II} + (\sum i :: \mathbf{1}) & \\
T' & = & \text{the carrier of the initial } \mathsf{F}\text{-algebra} & \\
\varphi_p(\varphi) & = & (\lambda k :: (m, t) \text{ where } (m, t) = \varphi m) & \\
\varphi_q(\varphi, \psi) & = & (\lambda k :: (m \min n, s +\!\!+ t) \text{ where } (m, s) = \varphi k,\ \ (n, t) = \psi k) & \\
\varphi_{r_i}() & = & (\lambda k :: (i, [k])) & \text{for all } i
\end{array}
$$

$$
\begin{array}{lcl}
\varphi & = & \varphi_p \triangledown \varphi_q \triangledown (\nabla i :: \varphi_{r_i}) \\
 & : & (\mathbb{N} \to \mathbb{N} \times \mathbb{N}*)_{\mathsf{F}} \to (\mathbb{N} \to \mathbb{N} \times \mathbb{N}*) \\
(\!|\varphi|\!) & : & T' \to (\mathbb{N} \to \mathbb{N} \times \mathbb{N}*)
\end{array}
$$

Notice that, as you can see from the equation for F, T' not only contains binary trees (as each parse trees is), but also trees in which a node has just one immediate constituent; these trees can not result from parsing by the underlying context-free grammar. See also the discussion just following Theorem (8).

References

[1] R.S. Bird. Using circular programs to eliminate multiple traversals of data. *Acta Informatica*, 21:239–250, 1984.

[2] M.M. Fokkinga. Tupling of catamorphisms yields a catamorphism. May 1990. CWI, Amsterdam.

[3] M.F. Kuiper and S.D. Swierstra. Using attribute grammars to derive efficient functional programs. In *Computing Science in The Netherlands*, pages 39–52, Stichting Informatica Onderzoek in Nederland, SION, CWI, Amsterdam, November 1987.

[4] M.F. Kuiper. *Parallel Attribute Evaluation*. PhD thesis, Utrecht University, 1989.