Inverse Currying Transformation on Attribute Grammars

Reinhard Wilhelm[*]

Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA 94720

ABSTRACT

Inverse currying transformation of an attribute grammar moves a context condition to places in the grammar where the violation of the condition can be detected as soon as the semantic information used in the condition is computed. It thereby takes into account the evaluation order chosen for the attribute grammar. Inverse currying transformations can be used to enhance context sensitive parsing using predicates on attributes, to eliminate sources of backtrack when parsing according to ambiguous grammars, and to facilitate semantics-supported error correction.

1. Motivation

Attribute grammars have found wide acceptance as a means to describe the static semantics of programming languages. Semantic attributes are used to carry scope, type and other declarative information; predicates on attribute occurrences in productions rule out programs, which are syntactically correct, but semantically meaningless, such as programs with missing or double declarations, type inconsistencies, or violations of parameter passing conventions. Seen formally, these "context conditions" restrict the language as defined by the context-free grammar to a subset satisfying semantic constraints of the language.

Example:
The production

$$ASSIGNMENT \rightarrow VAR := EXPR$$

might have the context condition

$$type(VAR) = type(EXPR)$$

or some less restrictive condition if the language allows automatic conversions. Hereby, it is assumed that synthesized attributes *type* are associated with the nonterminals in the VAR- and EXPR-subgrammars indicating the type of the

[*]on leave from: Universität des Saarlandes, Fachbereich 10-Informatik, D-6600 Saarbrücken, Fed. Rep. of Germany. Research reported herein was supported by a travel grant of the Deutsche Forschungsgemeinschaft and by grant MCS80-05144 from the National Science Foundation.

variable or the expression. There are other ways to describe the same context condition; but this seems to be the "natural" one. In particular, it does not contain any assumption about the order in which attributes are evaluated, i.e. the order in which the VAR- and EXPR-subtrees are traversed by the attribute evaluator.

If the evaluation order is known, e.g. left-to-right depth-first, one could transform the attribute grammar so that the VAR can tell the EXPR what types it may legally have. This might enhance the localization of semantic errors, as it allows for the detection of the smallest subexpression that causes the expression to have the wrong type.

Other interesting applications for this transformation lie in the area of context-sensitive parsing [JMJ80] [WaJ80] in particular when working with ambiguous grammars. The Graham-Glanville code generation scheme [GlG78] [Hen83] works with an LR-parser generated from a syntactically ambiguous grammar. Shift-reduce conflicts are resolved in favour of shifts to generate "larger" instructions; reduce-reduce conflicts are resolved by semantic constraints and/or cost criteria. A semantic constraint whose violation is discovered only after some reductions have been made and the corresponding instructions issued requires backup to find some other sequence of instructions previously ruled out in favour of a presumably better sequence.

Example:
The following productions describe two addressing modes and three instructions for some machine:

```
INDEX → + REG * const REG
  condition: value(const) ∈ {1,2,4,8}
       size(INDEX) := value(const)

ADDRESS → INDEX
     size(ADDRESS) := size(INDEX)

ADDRESS → REG

RVAL → indir ADDRESS
  condition: size(indir) = size(ADDRESS)

REG → * REG REG

REG → + REG REG

REG → const
```

When parsing the intermediate program representation

indir + REG * const REG

any reductions using the last two productions would be rejected in favour of the "larger" production given first, if the size of the const is one out {1,2,4,8}. This would reduce the string to

indir INDEX

But backup is necessary, if after reduction to

indir ADDRESS

the *size* condition of the production for RVAL is discovered to be violated. Again, as in the previous example, the *size* information at the indir node could be passed down the ADDRESS tree to discover the violation of the size condition upon shift instead of upon reduction.

The transformation of attribute grammars described in this paper would move semantic constraints to places in the grammar where their violation could be observed earlier. This eliminates sources of backup.

The transformation can also be used to exploit semantic information for the correction of syntax errors [Sch82]. The transformation was first used in [Wil79] to show different attribute associations with with different properties for the same language. We call the transformation *inverse currying* because of the related transformation in combinatory logic, which transforms a function into a functional by pulling out one of its argument domains. This was named after the logician Haskell Curry.

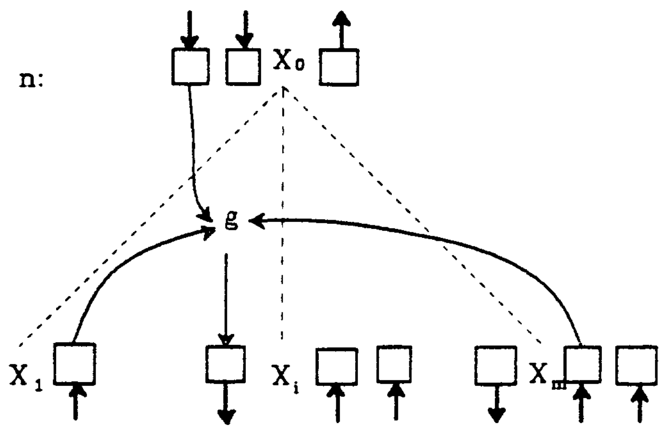
2. Terminology and Notation

An attribute grammar consists of a context-free grammar, called its *underlying* context-free grammar, an association of attributes with terminal and nonterminal symbols, and an extension of productions by semantic rules and context-conditions. The set A(X) of attributes associated with a a symbol X is the union of the disjoint sets I(X) and S(X) of *inherited* and *synthesized* attributes. Both an attribute a of symbol X and its occurrences will be denoted by $a(X)$. Occurrences of inherited attributes on the left side of a production n and of synthesized attributes on the right side of a production are called *importing* attribute occurrences, since at an instance of n in a syntax tree their instances import values from the context of that instance of the production. All other occurrences of attributes in productions are called *exporting*. For any exporting occurrence of an attribute b in a production n: $X_0 \rightarrow X_1 \cdots X_m$ there must be exactly one *semantic rule*

$$b(X_j) := g(a_1(X_{i_1}), \cdots, a_k(X_{i_k}))$$

determining how values for this attribute occurrence are computed. It is generally assumed that the arguments of g are either importing occurrences of attributes or constants. Also associated with a production may be *context-conditions* $p(a_1(X_{i_1}), \cdots, a_k(X_{i_k}))$, p being a k-ary boolean predicate, describing semantic constraints on the language defined by the underlying context-free grammar. We also assume that the arguments to p are all importing occurrences. Fig. 1 gives a graphical representation of an attributed production. Inherited attributes are represented by boxes to the left of symbol occurrences, synthesized attributes to the right. Only one semantic rule is depicted by function symbol g with arrows indicating flow from arguments and to the target. A context-condition is written below the production. Heavy arrows pointing to importing and heavy arrows pointing from exporting occurrences indicate the interface to the upper and lower context.

The transformation described in this paper involves manipulation of boolean predicates used as context-conditions and of functions used in semantic rules.



condition: $p(a(X_0), b(X_1), b(X_i))$

Fig. 1 An attributed production

Given a predicate

$$p : A_1 \times \cdots \times A_k \rightarrow \{ \mathit{true}\ , \mathit{false} \} .$$

If there exists $1 \leq r < k$ and nontrivial functions

$$f_r^p : A_1 \times \cdots \times A_{r-1} \rightarrow 2^{A_r},$$

$$\cdots$$

$$f_k^p : A_1 \times \cdots \times A_{r-1} \rightarrow 2^{A_k}$$

such that

$$p(a_1, \cdots, a_{r-1}, b_r, \cdots, b_k) = \mathit{true} \text{ iff } b_j \in f_j^p(a_1, \cdots, a_{r-1}) \text{ for all } j: r \leq j \leq k$$

then we call the f_j^p's *inverse curried functions* for the predicate p.
We call these functions *nontrivial* if they are not the constant function ϕ.

Suppose some arguments to a predicate p are given. The inverse curried functions for p compute for all the other argument positions the set of arguments that together with the given arguments will satisfy the predicate.

Given a function

$$g : A_1 \times \cdots \times A_k \rightarrow B$$

If there exist $1 \leq r < k$ and nontrivial functions

$$f_r^g : 2^B \times A_1 \times \cdots \times A_{r-1} \rightarrow 2^{A_r}$$

$$\cdots$$

$$f_k^g : 2^B \times A_1 \times \cdots A_{r-1} \rightarrow 2^{A_k}$$

such that for any $b \subseteq B$:

$$g(a_1, \cdots, a_{r-1}, c_r, \cdots, c_k) \in b$$

iff

$$c_j \in ff(b,a_1,\cdots,a_{r-1}) \text{ for all } j: r \le j \le k.$$

then we call the functions ff *inverse curried functions* for the function g.

In the following we will always use letters g,h for functions in general and letters $p,q,\ldots$ for boolean predicates. Hence it will always be clear how the actual curried inverse functions are constructed.

In general, the inverse curried functions might not be computable, and in case they are, it might be impractical to compute or represent them. But functions used in attribute grammars are mostly total functions on small domains.

3. The Transformation

The transformation is based on a left-to-right depth-first evaluation order as required for the evaluation of L-attributed grammars [LRS74], i.e. grammars with no right-to-left attribute dependencies. It removes context-conditions replacing them by functions computing the "legal domains" of attribute values guaranteeing the satisfaction of the removed conditions, functions propagating these domains down the tree, and membership tests at those places where attribute values are computed.

There will be three steps constituting this transformation:

Steps of type (1) remove a context condition from a production, associate new inherited (domain) attributes with nonterminals and semantic rules initializing these domain attributes with the production.

Steps of type (2) and (3) handle any such newly introduced domain attribute. They thus have two formal parameters, a synthesized attribute b and the corresponding inherited domain attribute i_b. (2) introduces functions propagating domains thereby inverting semantic functions wherever possible. If there is a context condition involving the attribute b under consideration, the combined requirements on the legal domains for b are propagated. If domains cannot be propagated, new context conditions are introduced into productions. They check whether an attribute computed in the production or associated with a terminal by the scanner is an element of the corresponding domain attribute. (3) takes care of occurrences of domain attributes without initialization.

The production n, which we will consider, syntactically always looks like

$$n:\ X_0 \rightarrow X_1 X_2 \cdots X_m$$

with the X_i being terminal or nonterminal symbols.

(1) Assume that connected with production $n:\ X_0 \rightarrow X_1 \cdots X_m$ there is a context condition

$$p(a_1(X_{i_1})),\ldots,a_k(X_{i_k})),\ i_j \in \{0,\ldots,m\} \text{ for } 1 \le j \le k.$$

To avoid some formal trouble we assume that $i_j \le i_{j+1}$ for $1 \le j \le k-1$ †.

Let domain(a_j)=A_j for $1 \le j \le k$. Thus p is a predicate

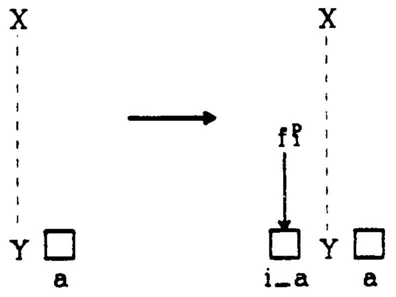
† This assumption reflects the left-to-right evaluation order, as the inverse curried functions are defined.

$p : A_1 \times \cdots \times A_k \rightarrow \{true, false\}$. If there exist inverse curried functions $f_r^p, \ldots, f_k^p$ for p, where we choose r minimal with $i_r \neq i_{r-1}$ *, we transform production n in the following way:

- delete p from production n;
- associate inherited (domain-) attributes $i\_a_r, \cdots, i\_a_k$ with the nonterminals among the $X_{i_r}, \cdots, X_{i_k}$;
- associate semantic rules $i\_a_j(X_{i_j}) := f_j^p(a_1(X_{i_1}), \cdots, a_{r-1}(X_{i_{r-1}}))$ for $r \leq j \leq k$ with production n, if X_{i_j} is a nonterminal;
- associate conditions $a_j(X_{i_j}) \in f_j^p(a_1(X_{i_1}), \cdots, a_{r-1}(X_{i_{r-1}}))$ with n, if X_{i_j} is a terminal.

When a syntax tree is decorated according to the new production n, any instance of an attribute i_a_j at an instance of n will contain all legal values for the corresponding instance of a_j , given values for the instances of attributes $a_1, \cdots, a_{r-1}$.

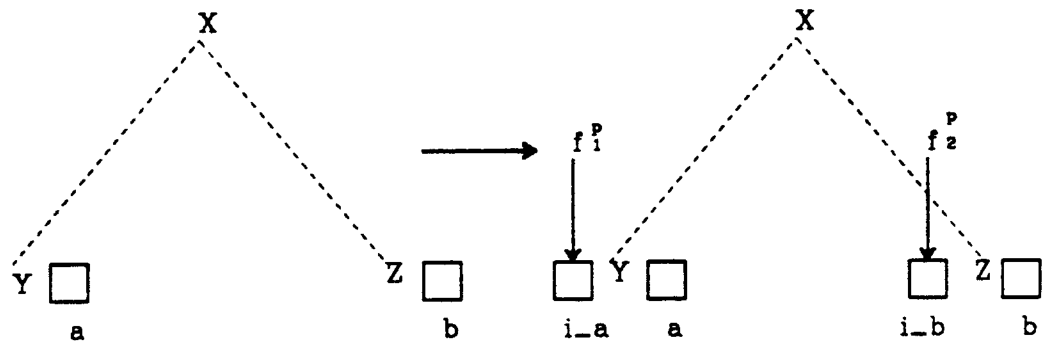
The above transformation covers the general case. In practice, there are only a few realistic cases. The most common of those are depicted in figures 1.1 - 1.5.



condition: p(a(Y))

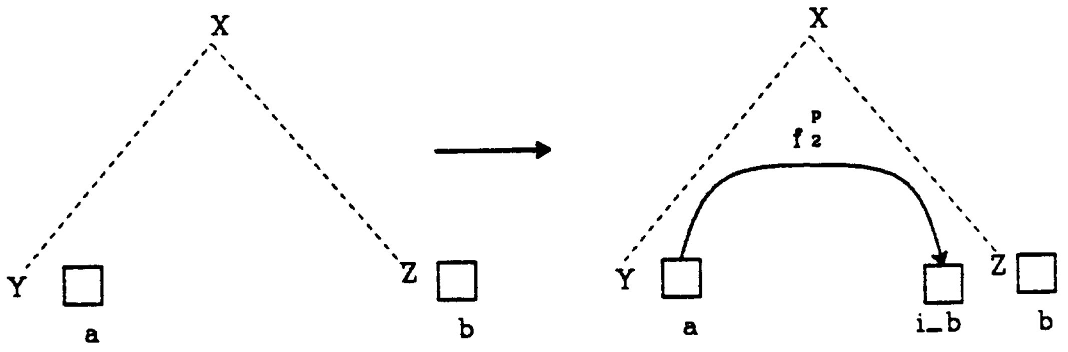
Fig. 1.1 A condition only depending on the syntactic context

* Choosing a minimal r corresponds to computing domains for attribute occurrences as far left in a right side as possible; the condition $i_r \neq i_{r-1}$ prevents the introduction of a dependency of an inherited domain attribute on a synthesized attribute at the same symbol occurrence.



condition: p(a(Y),b(Z))

Fig. 1.2 Another condition only depending on the syntactic context



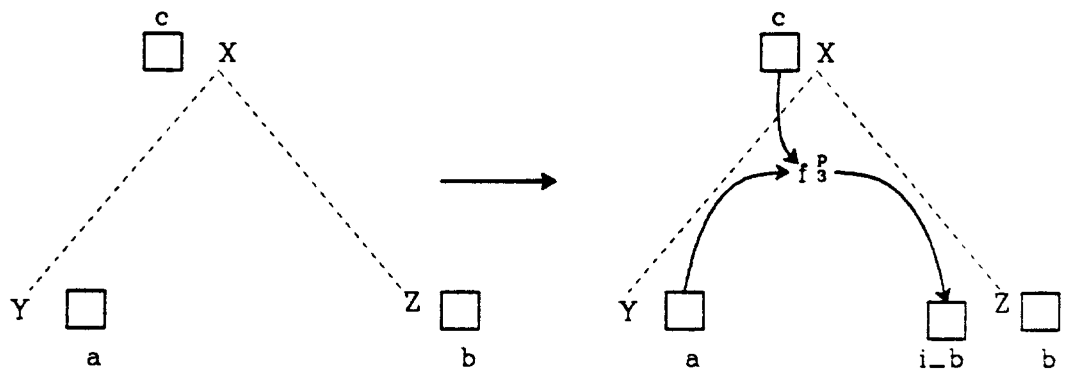
condition: p(a(Y),b(Z))

Fig. 1.3 Introducing left-to-right information flow

The legal domains must be propagated down the trees. Step (2) makes the necessary changes to the grammar.

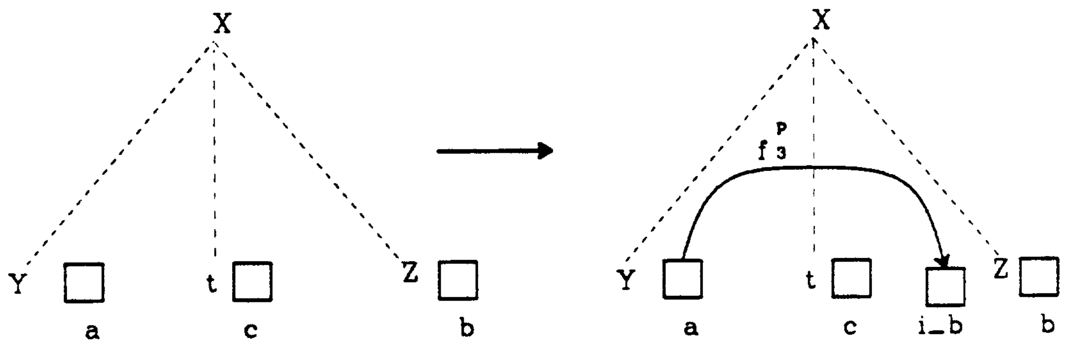
(2) Assume that there is a synthesized attribute b and a newly introduced inherited domain attribute i_b associated with X_0, and a semantic rule $b(X_0) := g(a_1(X_{i_1}), \cdots, a_k(X_{i_k}))$ associated with production n. Assume as in (1) that $i_j \leq i_{j+1}$ for $1 \leq j \leq k-1$.

(2.1)If inverse curried functions $f_r^g, \cdots, f_k^g$ exist (r minimal with $i_r \neq i_{r-1}$), then we can propagate legal attribute domains by the following changes to production n:
- associate inherited attributes i_a_j with the nonterminals among the $X_{i_r}, \cdots, X_{i_k}$;
- associate semantic rules $i\_a_j(X_{i_j}) := f_j^g(i\_b(X_0), a_1(X_{i_1}), \cdots, a_{r-1}(X_{i_{r-1}}))$ for $r \leq j \leq k$ with production n, if X_{i_j} is a nonterminal and $i_j \neq 0$;
- associate a condition $a_j(X_{i_j}) \in f_j^g(i\_b(X_0), a_1(X_{i_1}), \cdots, a_{r-1}(X_{i_{r-1}}))$ with n, if



condition: p(c(X),a(Y),b(Z))

Fig. 1.4 Left-to-right and top down information flow

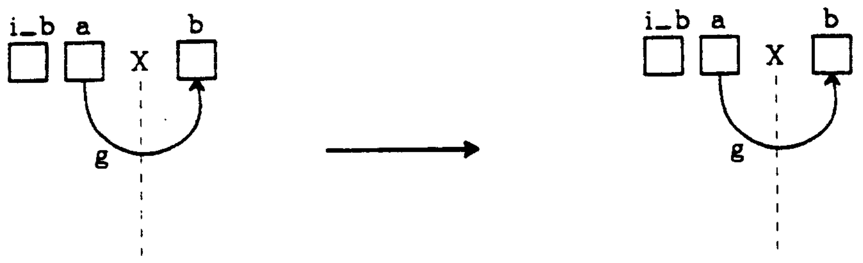


condition: p(a(Y),c(t),b(Z)) condition: c(t) $\in f_2^p$(a(Y))

Fig. 1.5 Testing an attribute of a terminal symbol

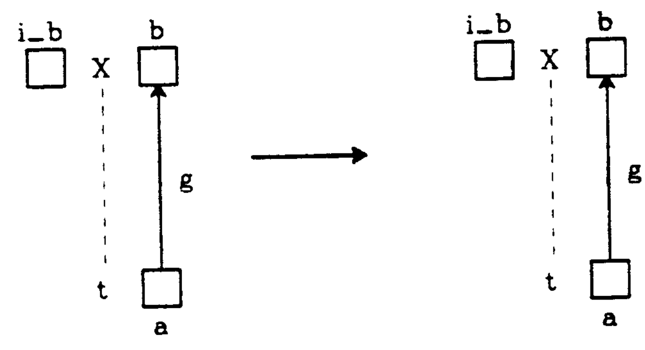
X_{i_j} is a terminal or i_j=0.

The most frequent examples of such changes are depicted in figures 2.1 - 2.4.



condition: $a(X) \in f_1^q(i\_b(X))$

Fig. 2.1 Introducing a membership test (here equivalent to g(a(X)) ∈ i_b(X))



condition: $a(t) \in f^{g}_{1}(i\_b(X))$

Fig. 2.2 Introducing a membership test for a terminal attribute

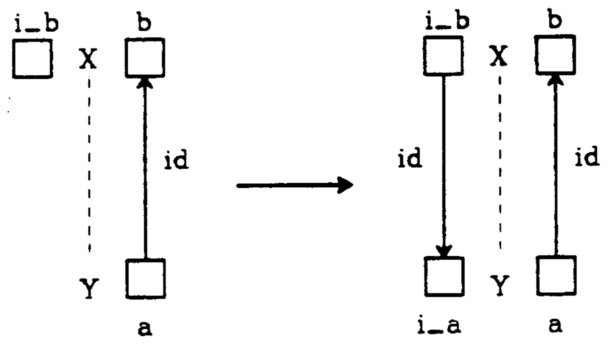
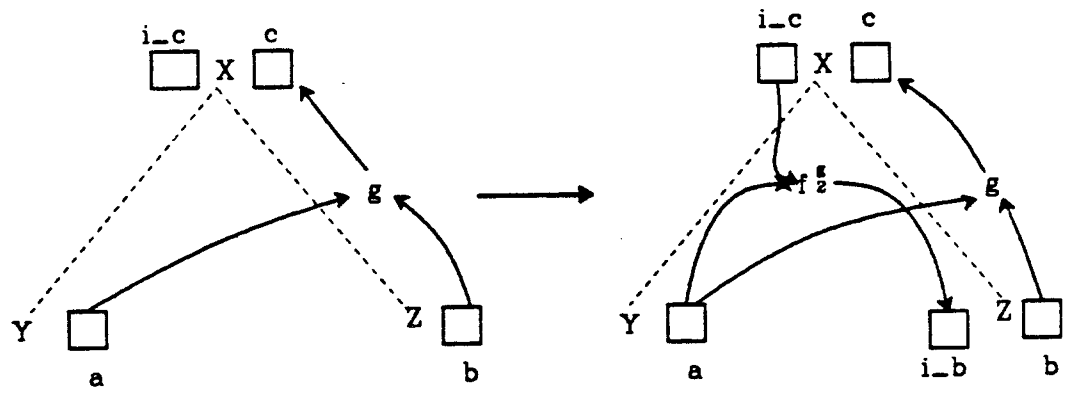


Fig. 2.3 The (common) case of identity transfer



condition: p(a(Y),b(Z))

Fig. 2.4 Left-to-right and top down information flow

(2.2)If no inverse curried functions exist, production n is changed as follows:
- Associate condition $g(a_1(X_{i_1}), \cdots, a_k(X_{i_k})) \in i\_b(X_0)$ with production n (cf. Fig. 2.5).

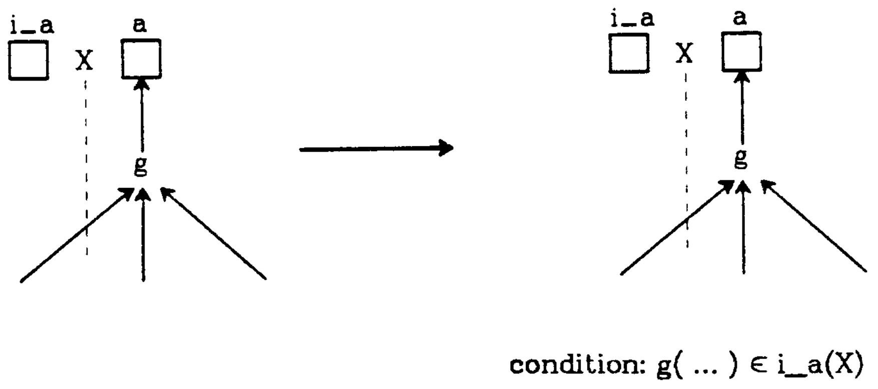


Fig. 2.5 No inverse curried function

(2.3)Another special case is the following:
$b(X_0)$ does not depend on any attribute occurrences but only on constants, i.e. $b(X_0) := g(c_1, \cdots, c_k)$. In this case we associate condition $g(c_1, \cdots, c_k) \in i\_b(X_0)$ with n (cf. Fig. 2.6).

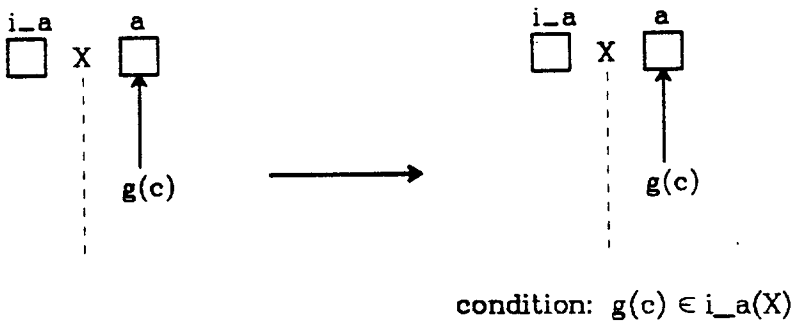
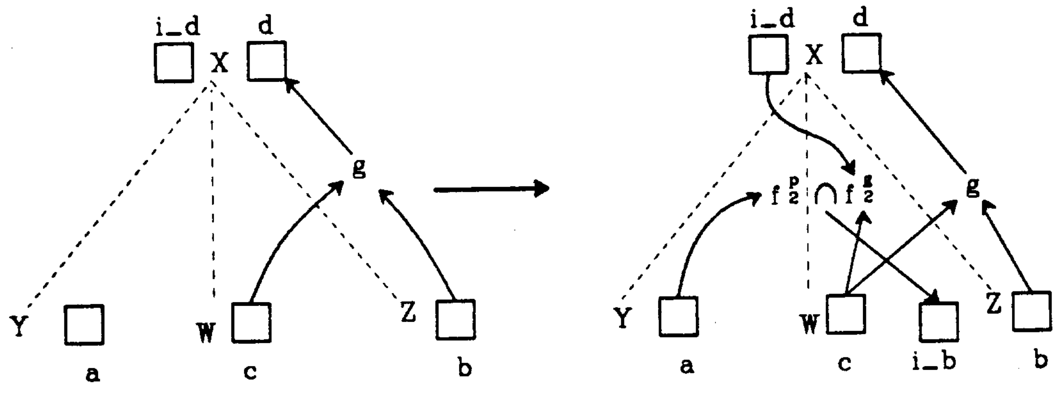


Fig. 2.6 Rule depending only on a constant

Steps (1) and (2) applied to the same production in any order may introduce several semantic rules $i\_b(X_j) := f^p(\cdots)$, $i\_b(X_j) := f^g(\cdots)$ and $i\_b(X_j) := f^h(\cdots)$ for the same occurrence of attribute i_b of X_j. If this is the case, then replace them by the semantic rule $i\_b(X_j) := f^p(\cdots) \cap f^g(\cdots) \cap f^h(\cdots)$ (cf. Fig. 2.7).

(3) If there are some exporting occurrences of domain attribute i_b without a semantic rule computing them, after all the above changes are made, then associate i_b( ) := domain(b) with the productions containing these occurrences. This may happen if some exporting occurrences of an attribute were involved in the transformation, but others were not.



condition: p(c(W),b(Z))

Fig. 2.7 Combined effect of a "global" condition (its domain in i_d) and a "local" condition (p)

4. Examples

Some productions from a grammar describing type calculation and type constraints for assignments, arithmetic and boolean expressions suffice to demonstrate most of the interesting cases occurring in inverse currying transformations. Different occurrences of the same symbol in one production are numbered from left to right starting with 1.

ASSIGNMENT → TARGET := SOURCE
condition: type(TARGET) = type(SOURCE)

Step (1) , special case from Fig. 1.3 introduces
i_type(SOURCE) := {type(TARGET)}

SOURCE → RELATION
type(SOURCE) := bool

Step (2.3) (Fig. 2.6) introduces
condition: bool $\in$ i_type(SOURCE)

SOURCE → EXPR
type(SOURCE) := type(EXPR)

Step (2), special case from Fig. 2.3, introduces
i_type(EXPR) := i_type(SOURCE)

EXPR → EXPR + TERM
condition: type($EXPR_2$) $\in$ {int,real} and
type(TERM) $\in$ {int,real} and
type($EXPR_2$) = type(TERM)

Step (2), special case from Fig. 2.7, introduces
i_type($EXPR_2$) := i_type($EXPR_1$)
i_type(TERM) := i_type($EXPR_1$) $\cap$ {type($EXPR_2$)} $\cap$ {int,real}

TERM → TERM ÷ FACTOR
condition: type($TERM_2$) = int and
type(FACTOR) = int
type($TERM_1$) := int

Steps (1) and (2), special cases from figures 1.2 and 2.6, introduce
i_type($TERM_2$) := {int}
i_type(FACTOR) := {int}
condition: int $\in$ i_type($TERM_1$)

FACTOR → id
type(FACTOR) := type_lookup(symbol(id))

Step (2),special case from Fig. 2.5, introduces
condition: type_lookup(symbol(id)) $\in$ i_type(FACTOR)

5. Invariance of the language under the grammar transformation

Let G_1 be an attribute grammar, G_2 the attribute grammar resulting form the inverse currying transformation applied to G_1. Since the transformation does not change the syntactic part of productions, both G_1 and G_2 have the same underlying context-free grammar G. Let us define the language described by context-free grammar G, L(G), as the set of its syntax trees. The language of attribute grammar G_i , L(G_i), is then defined as the set of trees t $\in$ L(G), such that all context-conditions of G_i are satisfied when decorating t according to G_i. The main result of this section states that G_1 and G_2 describe the same language, i.e. the inverse currying transformation leaves the defined language invariant. This is the claim of Theorem 1. Theorem 2 states that the transformation is compatible with a left-to-right evaluation strategy.

Theorem 1: $L(G_1) = L\ (G_2)$, i.e. the inverse currying transformation leaves the defined language invariant.

Proof:

Let us consider a syntax tree t of the context-free grammar underlying both G_1 and G_2. Let t_1 (t_2) be the resulting attributed tree after decoration of t according to G_1 (G_2).

We prove the theorem by showing that both $t \in L\ (G_1)$ and $t \in L(G_2)$ are equivalent to the following claim:

(B) For all nonterminal symbols X and for all $b \in S(X)$ (in AG G_1):
If the transformation introduced an inherited attribute i_b for X (in AG G_2), then the value of any instance of b in t_1 is an element of the value of the corresponding instance of i_b in t_2 (cf. Fig. 3) and any attribute of a terminal involved in an introduced membership test satisfies it.

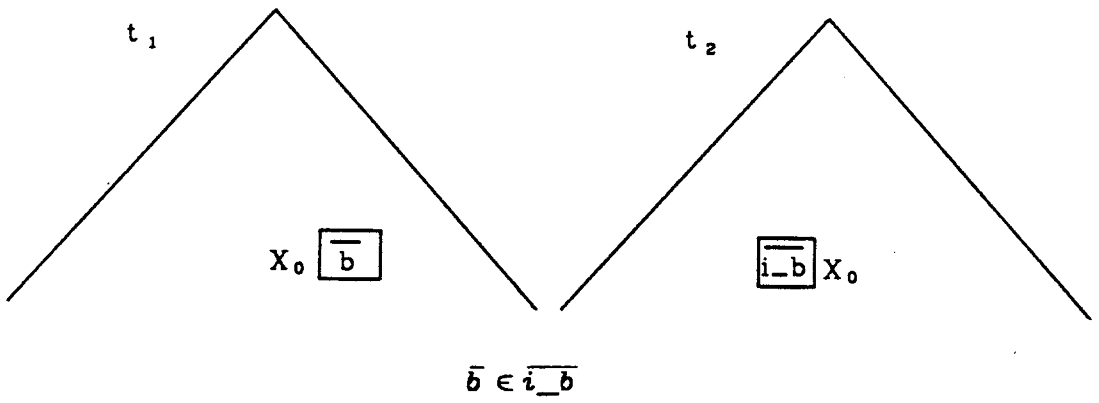


Fig. 3

The proof of '(B) <= t ∈ $L(G_1)$' uses top down tree induction on t_1 and t_2, the proof of '(B) <= t ∈ $L(G_2)$' uses bottom up tree induction on t_1 and t_2.

Proof of '(B) <=> t ∈ $L(G_1)$':

'=>'

It has to be shown that all instances of context-conditions are satisfied in t_1. Select an arbitrary instance of production n: $X_0 \rightarrow X_1 \cdots X_m$ with context-condition $p(a_1(X_{i_1}), \cdots, a_k(X_{i_k}))$ in G_1. Transformation step (1) introduces inherited attributes $i\_a_r, \cdots, i-a_k$ for the nonterminals among the $X_{i_1}, \cdots, X_{i_k}$, resp., and semantic rules $i\_a_j(X_{i_j}) := f_j^p(a_1(X_{i_1}), \cdots, a_{r-1}(X_{i_{r-1}}))$, $r \le j \le k$.

(B) states that the values of the a_j's in t_1 (at this instance of the production) are members of the values of the i_a_j's in t_2 (at the corresponding instance of the production) and that the terminal attributes satisfy the required membership tests in t_2. From the construction of the f_j^p and these membership tests it follows that the context-condition p is satisfied at this instance of the production.

'<=' top down tree induction

Select an instance of production n: $X_0 \rightarrow X_1 \cdots X_m$ in t. Assume there is a condition $p(a_1(X_{i_1}), \cdots, a_k(X_{i_k}))$ associated with n, which is satisfied according to the assumption t $\in L(G_1)$. Transformation step (1) introduced inherited attributes $i\_a_r, \cdots, i\_a_k$ for the nonterminals among the $X_{i_r}, \cdots, X_{i_k}$ and semantic rules $i\_a_j(X_{i_j}) := f_j^p(a_1(X_{i_1}), \cdots, a_{r-1}(X_{i_{r-1}}))$ with n.
Since $p(a_1(X_{i_1}), \cdots, a_k(X_{i_k}))=true$ it follows from the construction of the f_j^p's that $\bar{a}_j \in \overline{i\_a_j}$ $(r \le j \le k)$, where $\bar{a}_j$ is the value of the regarded instance of a_j in t_1, and $\overline{i\_a_j}$ is the value of the regarded instance of i_a_j in t_2.

The claim is trivially true if an instance of the new inherited attribute i_b was initialized with the full domain of the corresponding synthesized attribute b.

This finishes the proof of the induction base.

For the induction step we regard an instance of nonterminal X_0 in t.
Let X_0 have (in G_1) a synthesized attribute b, and (in G_2) an inherited attribute i_b introduced by the inverse currying transformation. The induction hypothesis says that the value $\bar{b}$ of b at this instance of X_0 in t_1 is a member of the value $\overline{i\_b}$ of i_b at the corresponding instance of X_0 in t_2. It has to be shown that the same holds for all pairs (c,i_c) of attributes at all children of this instance of X_0 related to each other by the transformation of section 2, where b depends on c (and thus i_c depends on i_b). Since b depends on c, there must be a semantic function $b(X_0) := g(a_1(X_{i_1}), \cdots, a_k(X_{i_k}))$ belonging to the production applied at this node in the tree. If g has inverse curried functions, then transformation step (2.1) has introduced inherited attributes $i\_a_r, \cdots, i\_a_k$ for the nonterminals among the $X_{i_r}, \cdots, X_{i_k}$ and semantic rules $i\_a_j(X_{i_j}) := f_j^g(i\_b(X_0), a_1(X_{i_1}), \cdots, a_{r-1}(X_{i_{r-1}}))$ $(r \le j \le k)$. By definition of the f_j^g using the induction hypothesis it follows that the values $\bar{a}_j$ of the instances of $a_j(X_{i_j})$ are members of the values $\overline{i\_a_j}$ of the instances of the $i\_a_j(X_{i_j}))$ for all j $: r \le j \le k$.

The claim still holds if another context-condition interferes with the downwards propagation of a domain, since this condition must also be satisfied, and since the conjunction of the two conditions translates into the intersection of the domains of legal values.

The proof of '(B) <=> t $\in L(G_2)$' is omitted since it is largely analogous to the first part.

Theorem 2: If G_1 is L-attributed, then G_2 is L-attributed.

Proof:

Transformation step (1) replaced a context-condition $p(a_1(X_{i_1}), \cdots, a_k(X_{i_k}))$ by a set of semantic rules $i\_a_j(X_{i_j}) := f_j^p(a_1(X_{i_1}), \cdots, a_{r-1}(X_{i_{r-1}}))$ $r \le j \le k$.
The assumptions $i_l \le i_{l+1}$ for $1 \le l \le k-1$ and $i_r \ne i_{r-1}$ imply $i_j > i_l$ for all $1 \le l \le r-1$ and $r \le j \le k$.
This means that all the newly introduced domain attributes $i\_a_r(X_{i_r}), \cdots, i\_a_k(X_{i_k})$ only depend on inherited attributes of X_0 and on synthesized attributes of X_{i_j} $(i_j \ne 0)$ to the left of any of the $X_{i_r}, \cdots, X_{i_k}$. Hence if there were no right-to-left dependences in the production before it was transformed, there will be none afterwards.

Transformation step (2) is handled with exactly the same argument.

Step (3) introduces only attribute initializations by constants which introduce no dependencies at all.

Altogether this proves the claim.

The following more general statement could be proved similarly:
Given a *simple multi-pass* attribute grammar, i.e. attribute evaluation can be done in a fixed number of left-to-right or alternating passes, and the instances of all occurrences of an attribute of a symbol are evaluated in the same pass. If the inverse currying transformation is applied to the attributes of left-to-right passes, it will not change the evaluation scheme for the grammar. One could also define an analogous transformation for right-to-left evaluation and apply it to right-to-left passes obtaining an analogous invariance result.

6. Practicality considerations

There are two questions concerning the practicality of the transformation. The first question is, who actually performs the transformation? Since the functions used in semantic rules are user supplied (function) procedures in some programming language, it is him who has to supply the inverse curried versions, too. But in order to estimate the amount of additional work thereby required, one must take into consideration, that only the functions have to be supplied, not the semantic rules in which they occur. Hence, the identity function on some attribute domain has to be inverted and not its many occurrences in the grammar.

The second question concerns the size growth of the attribute grammar. Experience with attribute grammars shows that very often a context condition "consumes" an attribute, i.e. an attribute value is computed somewhere in the tree and transferred to the instance of the production only for the purpose of context checking. This attribute may become "dead", when the context condition is removed from the grammar. In this case the attribute and the semantic rules computing and transferring it can be deleted from the grammar after the transformation has been applied.

7. Conclusion

An equivalence transformation on attribute grammars was defined. It allows the describer to separate between the language he wants to define and the behaviour of the acceptor generated from the description. Thus he may place context conditions at productions where they "naturally" belong, while the acceptor may expose a desired behaviour, e.g. sophisticated error diagnosis, less backtracking or better use of available semantic information for syntax error recovery.

Inverse currying transformations, like many other algorithms on attribute grammars, suffer from the lack of knowledge about the meaning of semantic functions and conditions. Their semantics can not be automatically determined, if they are written in a general purpose programming language like PASCAL or C. This problem will be eased, once special purpose languages will be designed for the description of compiler tasks such as identification of identifiers, code generation and code improvement. This will make other equivalence transformations on attribute grammars possible massaging them into different forms for different purposes.

Concluding, I would like to mention two extensions to the transformation as described in this paper. The following generalization of the definition of inverse curried function was proposed by Eduardo Pellegri.

Given a predicate

$$p : A_1 \times \cdots \times A_k \rightarrow \{ \mathit{true} \ , \mathit{false} \ \}.$$

The definition from section 2 determines an index r, such that given values of the first r-1 arguments, one can compute domains for the last k-r arguments. The generalization would require that there exist a strictly monotone increasing sequence of such r's: $r_1, \cdots, r_l$ and functions

$$f^p_{r_1} : A_1 \times \cdots \times A_{r_1 -1} \rightarrow 2^{A_{r_1}}$$

$$\cdots$$

$$f^p_{r_2 -1} : A_1 \times \cdots \times A_{r_1 -1} \rightarrow 2^{A_{r_2 -1}},$$

$$f^p_{r_2} : A_1 \times \cdots \times A_{r_2 -1} \rightarrow 2^{A_{r_2}}$$

$$\cdots$$

$$f^p_{r_3 -1} : A_1 \times \cdots \times A_{r_2 -1} \rightarrow 2^{A_{r_3 -1}},$$

$$\cdots$$

$$f^p_{r_l} : A_1 \times \cdots \times A_{r_l -1} \rightarrow 2^{A_{r_l}}$$

$$\cdots$$

$$f^p_k : A_1 \times \cdots A_{r_l -1} \rightarrow 2^{A_k},$$

such that

$$p(a_1, \cdots, a_k) = \mathit{true}$$

iff

$$a_{r_j + i} \in f^p_{r_j + i}(a_1, \cdots, a_{r_j -1})$$

for all $1 \le j \le l$ and $0 \le i \le r_{j+1} - r_j - 1$.

This generalization might catch some additional cases, although experience shows that right sides of productions tend to be rather short.

Another extension was proposed by Ulrich Möncke. Inverse curried functions do not exist, if it is impossible to tell different attributes independantly, which values they'd better have. In some cases, it may be possible to compute a cartesian product as the legal domain for several attributes of one occurrence of a nonterminal.

Acknowledgements
I would like Phil Garrison, Robert Henry, Ulrich Möncke, Eduardo Pellegri, and Beatrix Weisgerber for careful reading of previous versions of this paper and many helpful suggestions.

References

[GlG78] R. S. Glanville and S. L. Graham, A New Method for Compiler Code Generation, *Proc. 5th ACM Symp. Principles of Programming Languages*, Tuscon, Ariz., January, 1978.

[Hen83] R. R. Henry, Experience with Practical Graham-Glanville Codegenerators, Ph.D. Thesis, Computer Sciences Division, UC Berkeley, 1983.

[JMJ80] N. D. Jones and M. Madsen, Attribute-Influenced LR Parsing, in *Semantics Directed Compiler Generation*, N. D. Jones, (ed.), Springer LNCS, 1980, 393-407.

[LRS74] P. M. Lewis, D. J. Rosenkrantz and R. E. Stearns, Attributed Translations, *Journal of Computer and System Sciences 9*, (1974), 279-307.

[Sch82] C. Schmauch, *Attribute Evaluation after Recovery from Syntax Error*, Fachbereich Informatik, Universität Kaiserslautern, 1982.

[WaJ80] D. Watt, Rule Splitting and Attribute-Directed Parsing, in *Semantics Directed Compiler Generation*, N. D. Jones, (ed.), Springer Verlag, 1980.

[Wil79] R. Wilhelm, Attributierte Grammatiken, *Informatik Spektrum*, 1979.