K-VISIT ATTRIBUTE GRAMMARS

by

Hanne Riis
Sven Skyum

DAIMI PB-121
June 1980

Computer Science Department
AARHUS UNIVERSITY
Ny Munkegade – DK 8000 Aarhus C – DENMARK
Telephone: 06 – 12 83 55
k-VISIT ATTRIBUTE GRAMMARS

Hanne Riis
Sven Skyum*

Computer Science Department
Aarhus University
Ny Munkegade
DK-8000 Aarhus C
Denmark

* from August 1 at:
Computer Science Department
University of Edinburgh
JCMB
The King's Buildings
Mayfield Road
Edinburgh EH9 3JZ
Scotland
Abstract

It is shown that any well-defined attribute grammar is k-visit for some k. Furthermore it is shown that given a well-defined grammar G and an integer k, it is decidable whether G is k-visit. Finally we show that the k-visit grammars specify a proper hierarchy with respect to translations.
1. INTRODUCTION

Knuth introduced attribute grammars as a formalism for associating meanings with strings of context-free languages ([7]). Because of the generality of the formalism there have been defined several subclasses of attribute grammars, which are often easier to handle (see e.g. [1], [5], [6], and [4]).

One of these subclasses is the one-visit (or re-ordered, [8]) attribute grammars of [2].

We define an extension of this class – the class of k-visit attribute grammars. The definition of a k-visit attribute grammar is a generalization of that of a one-visit attribute grammar: An attribute grammar $G$ is k-visit if for any derivation tree $t$ of $G$ it is possible to evaluate all the attributes associated with $t$ by walking through $t$ in such a way that no node in $t$ is visited more than $k$ times.

While it is possible to find a local property for the productions, which is equivalent to the global one-visit property (see [2]), there seems to be no such local property corresponding to the k-visit property for an attribute grammar.

We show in this paper that any well-defined (see [7]) attribute grammar $G$ is k-visit for some $k$. Furthermore it is shown using a pumping argument, that given a well-defined grammar $G$ and an integer $k$ it is decidable whether $G$ is k-visit. Thus we can effectively for any well-defined attribute grammar $G$ find the minimal $k$ such that $G$ is k-visit. Finally we show that the k-visit attribute grammars specify a proper hierarchy with respect to translations.
2. NOTATION AND DEFINITIONS

In this section we will recall the definition of an attribute grammar and define some related concepts. We use a definition of an attribute grammar similar to that given in [2].

A semantic domain \( \mathcal{D} \) is a pair \((\mathcal{D}, \mathcal{F})\) where \( \mathcal{D} \) is a set of sets and \( \mathcal{F} \) is a set of mappings of functionality \( \mathcal{D}_1 \times \mathcal{D}_2 \times \ldots \times \mathcal{D}_m \rightarrow \mathcal{D} \) \( (m \geq 0) \), where \( \mathcal{D} \) and \( \mathcal{D}_j \) for \( 1 \leq j \leq m \) are sets in \( \mathcal{D} \). The elements of \( \mathcal{D} \) will be called domains whereas the elements of \( \mathcal{F} \) will be called semantic functions.

An attribute grammar (AG) \( G \) over a semantic domain \( \mathcal{S} = (\mathcal{D}, \mathcal{F}) \) can be considered as an extension of a context-free grammar (CFG) \( G_u = (N, T, P, Z) \).

To each nonterminal \( X \) of \( G_u \) there is associated a fixed set \( A(X) \) called the attributes of \( X \). The set \( A(X) \) is divided into two disjoint sets, \( I(X) \), the set of inherited attributes, and \( S(X) \), the set of synthesized attributes.

To each element in \( A(X) \) there is associated a domain from \( \mathcal{D} \). We require without loss of generality that \( S(Z) \) is a singleton and that \( I(Z) = \emptyset \).

To each production \( p: F ::= w_0 F_1 w_1 F_2 w_2 \ldots w_{n-1} F_n w_n, \) \( w_j \in T^* \) \( (0 \leq j \leq n) \), \( F_j \in N \) \( (1 \leq j \leq n) \) of \( G_u \) there is associated so-called semantic rules.

There will be exactly one semantic rule for each attribute \( b \) in the set \( S(F) \cup I(F_1) \cup I(F_2) \cup \ldots \cup I(F_n) \). A semantic rule for \( b \) consists of a semantic function \( f: \mathcal{D}_1 \times \mathcal{D}_2 \times \ldots \times \mathcal{D}_m \rightarrow \mathcal{D} \) from \( \mathcal{F} \) together with \( m \) attributes \( a_1, a_2, \ldots, a_m \) from the set \( I(F) \cup S(F_1) \cup S(F_2) \cup \ldots \cup S(F_n) \). The attributes \( b \) and \( a_j \) for \( 1 \leq j \leq m \) must satisfy that the domain \( \mathcal{D} \) is associated with \( b \) whereas the domain \( \mathcal{D}_j \) is associated with \( a_j \) for \( 1 \leq j \leq m \). We say that \( b \) depends on \( a_j \) for \( 1 \leq j \leq m \).

**Example 2.1**
Let us define an AG \( G \) over a semantic domain \( \mathcal{S} = (\mathcal{D}, \mathcal{F}) \).

\( \mathcal{D} \) will consist of the single set \( \mathcal{D} = \{A, B\}^*\), i.e. sequences of \( A \)'s and \( B \)'s.

In \( \mathcal{F} \) we have five semantic functions:
The underlying grammar $G_u$ of the AG $G$ will be

\[
\begin{align*}
Z & ::= X \\
X & ::= X X | Y \\
Y & ::= A | B
\end{align*}
\]

To each attribute of any symbol we associate the domain $D$. We give the productions of the AG and their semantic rules below in an affix-like notation. The inherited attributes will be prefixed by downward arrows (↓) whereas synthesized attributes will be prefixed by upward arrows (↑).

\[
\begin{align*}
p_1: \quad <Z \uparrow \iota(a)> & ::= <X \downarrow \lambda(\, ) \uparrow a> \\
p_2: \quad <X \downarrow a \uparrow \iota(c)> & ::= <X \downarrow \iota(b) \uparrow c> <X \downarrow \iota(a) \uparrow b> \\
p_3: \quad <X \downarrow a \uparrow \&(a,b)> & ::= <Y \uparrow b> \\
p_4: \quad <Y \uparrow \lambda(\, )> & ::= A \\
p_5: \quad <Y \uparrow \lambda(\, )> & ::= B
\end{align*}
\]

To each node in a derivation tree $t$ defined by the CFG $G_u$ we will associate attributes similar to those of the symbol labeling the node. Let $n$ be a node labeled with a nonterminal $X$ in $t$. If $p: X ::= X_1 X_2 \ldots X_m$ is the production used to expand $n$ in $t$ then the semantic rules for the attributes in $S(X)$ associated with $p$ are used to give values to the synthesized attributes of $n$ (denoted $S(n)$). On the other hand if $X$ occurs on the right hand side of the production $q: Y ::= Y_1 Y_2 \ldots Y_h$ and $n$ is introduced in $t$ by an application of $q$ then the semantic rules for the attributes in $I(X)$ associated with $q$ are used to give values to the inherited attributes of $n$ (denoted $I(n)$).
In the following a derivation tree will have root labelled $Z$ and leaves labelled by symbols from $T$.

If $t$ is a derivation tree and $n$ a node in $t$ then $t^n$ denotes the subtree of $t$ with root $n$, and $t^n$ denotes the tree $t$ with the subtree of $n$, except $n$ itself, removed.

We want to give values to (or evaluate) all the attributes associated with (the nodes of) the tree $t$. This is done during a walk through the tree. When arriving at a node we may evaluate some of the inherited or synthesized attributes of the node and after that continue to either the father, a son or a brother of the node or the node itself.

A walk through a tree $t$ is a sequence

$$s = \langle n_1, A_1 \rangle \langle n_2, A_2 \rangle \cdots \langle n_r, A_r \rangle$$

where

1) $n_j$ is an interior node in $t$ for $1 \leq j \leq r$
2) If $n_j \neq n_{j+1}$ then $n_{j+1}$ is the father, a son or a brother of $n_j$, $1 \leq j < r$
3) $A_j$ is a subset of the inherited or synthesized attributes of $n_j$ for $1 \leq j \leq r$

For each node $n$ in $t$ we define

$$s(n) = A_{i_1} A_{i_2} \cdots A_{i_m}$$

if $i_1 < i_2 < \ldots < i_m$ and $\{ i_j : 1 \leq j \leq m \} = \{ h : n_h = n, 1 \leq h \leq r \}$
A computation sequence for a derivation tree $t$ is a walk
\[ cs = \langle n_1, A_1 \rangle \langle n_2, A_2 \rangle \ldots \langle n_r, A_r \rangle \]
through $t$ satisfying

4) $n_1 = n_r$ is the root of $t$

5) the attributes of $A_j$ do not depend on those of $A_i$ for $i > j$, $1 \leq j \leq r$

6) for each interior node $n^l$ in $t$:
   a) $B_j \subseteq l(n^l)$ if $j$ is odd, $1 \leq j \leq h$
   b) $B_j \subseteq s(n^l)$ if $j$ is even, $1 \leq j \leq h$
   c) $\bigcup_{j=1}^{h} B_j = l(n^l) \cup s(n^l)$
   d) $B_i \cap B_j = \emptyset$ if $i \neq j$
   e) let $cs = cs_1 \langle n^l, B_j^l \rangle \langle n^l_1, C_1 \rangle \langle n^l_2, C_2 \rangle \ldots$
      $\langle n^l_k, C_k \rangle \langle n^l_1, B_{j+1}^l \rangle cs_2$;
      if $j$ is odd (even) then $n_i$ is a node in $t_{n_i} (t^n)$ for $1 \leq i \leq k$

(Note, that 4) implies that $h$ is even.)

Thus when walking 'down' in a subtree we evaluate some of the inherited attributes of the root of the subtree. When returning from the subtree we evaluate some of the synthesized attributes of the root of the subtree.
Example 2.2
Consider the following derivation tree $t$ of the AG of Example 2.1.

Let $I$ denote the inherited attributes of a node and $S$ the synthesized.
A computation sequence for $t$ which traverses $t$ twice from left to right is

1. pass
   \[
   \begin{align*}
   \langle n_1, \emptyset \rangle & \langle n_2, I \rangle \langle n_3, \emptyset \rangle \langle n_4, \emptyset \rangle \langle n_4, S \rangle \langle n_3, \emptyset \rangle \\
   \langle n_5, I \rangle & \langle n_6, \emptyset \rangle \langle n_6, S \rangle \langle n_5, S \rangle \langle n_2, \emptyset \rangle \langle n_1, \emptyset \rangle
   \end{align*}
   \]

2. pass
   \[
   \begin{align*}
   \langle n_1, \emptyset \rangle & \langle n_2, \emptyset \rangle \langle n_3, I \rangle \langle n_4, \emptyset \rangle \langle n_4, \emptyset \rangle \langle n_3, S \rangle \\
   \langle n_5, \emptyset \rangle & \langle n_6, \emptyset \rangle \langle n_6, \emptyset \rangle \langle n_5, \emptyset \rangle \langle n_2, S \rangle \langle n_1, S \rangle
   \end{align*}
   \]

Let $t$ be a derivation tree and $n$ an interior node in $t$. Consider a computation sequence $cs$ for $t$ of the form

\[
cs = cs_1 \langle n, B_1 \rangle cs_2 \langle n, B_2 \rangle cs_3
\]

where $|cs_1(n)|$ (the length of $cs_1(n)$) is even and $cs_2(n) = \lambda$ (the empty string).

The subsequence $cs_2$ will walk through the subtree $t_n$ of $t$ with root $n$. We say that $\langle n, B_1 \rangle cs_2 \langle n, B_2 \rangle$ specifies a visit to $t_n$ and that the node $n$ is visited once. Thus if $|cs(n)| = 2h$ then $n$ is visited $h$ times.

$t$ is said to be $k$-visit if there exists a computation sequence $cs$ for $t$ satisfying that for each node $n$ in $t$: $|cs(n)| \leq 2k$. 

\[\square\]
An AG $G$ is $k$-visit if any derivation tree of $G_u$ is $k$-visit.

Knuth introduced a subclass of AGs called well-defined AGs ([7]). If an AG is well-defined then the attributes do not depend circularly on each other in any derivation tree. The test for the well-definedness property of an AG has exponential time complexity (see [3]). It is easy to show that if the AG is well-defined then there exists a computation sequence for any derivation tree.

Let $t$ be a derivation tree of a well-defined AG. Given a computation sequence for $t$ we can determine the value of the attribute at the root of $t$. This value will be denoted the meaning of $t$ in the semantic domain $\mathcal{A}$: meaning $(t, \mathcal{A})$. This value is independent of the choice of computation sequence (see [9]).

The translation $T(G, \mathcal{A})$ specified by the well-defined AG $G$ is defined by

$$T(G, \mathcal{A}) = \{ (t, \text{meaning}(t, \mathcal{A})) : t \text{ is a derivation tree for } G_u \}$$

**Example 2.3**
For the derivation tree $t$ of Example 2.2 we have meaning$(t, \mathcal{A}) = BA$.

The AG of Example 2.1 specifies the translation

$$T(G, \mathcal{A}) = \{ (t, v^R) : t \text{ is a derivation tree for } v \text{ in } G_u \}$$

($v^R$ is the string $v$ reversed.)
3. THE k-VISIT PROPERTY

In this section we will show that for any well-defined AG G there exists a k such that G is k-visit. This is done in two steps. First we show that any computation sequence can be transformed into a special kind of computation sequence called a reduced computation sequence. In the second step we use these reduced computation sequences to determine a k such that the AG is k-visit.

Definition 3.1
A reduced computation sequence for a derivation tree t is a computation sequence cs for t satisfying:

if n is an interior node in t and

\[ cs = cs_1 \langle n, \emptyset \rangle cs_2 \langle n, \emptyset \rangle cs_3 \]

where \(|cs_1(n)|\) is even and \(cs_2(n) = \lambda\)
then \(cs_1(n) = cs_3(n) = \lambda\). □

Intuitively, if cs is a reduced computation sequence then we will only visit a node n when some of its attributes can be evaluated. And if there are no attributes associated with n then we will visit n at most once.
Note that a reduced computation sequence need not be the shortest computation sequence for a tree.

Example 3.2
A reduced computation sequence for the tree in Example 2.2 is:

\[ \langle n_1, \emptyset \rangle \langle n_2, \lambda \rangle \langle n_5, \lambda \rangle \langle n_6, \emptyset \rangle \langle n_6, S \rangle \langle n_5, S \rangle \]
\[ \langle n_3, \lambda \rangle \langle n_4, \emptyset \rangle \langle n_4, S \rangle \langle n_3, S \rangle \langle n_2, S \rangle \langle n_1, S \rangle \]

□

Any computation sequence for a derivation tree t can be transformed into a reduced computation sequence. To see this we will define two transformations \(T_1\) and \(T_2\) on computation sequences.

The purpose of both of these transformations will be to remove what we might call superfluous visits to subtrees.
The transformation $T_1$ can be applied to a computation sequence $cs$ of the form

$$cs = cs_1 <n, \emptyset> cs_2 <n, \emptyset> cs_3 <n, B> cs_4$$

where $|cs_1(n)|$ is even and $cs_2(n) = cs_3(n) = \lambda$.

Thus there will be a visit to $n$ where no attributes of $n$ become evaluated and there will be at least one more visit to $n$ (note that $B$ might be empty). The result of applying $T_1$ to $cs$ is the sequence

$$cs' = cs_1 cs_3 <n, B> cs_2 cs_4$$

It can be proved that $cs'$ is a computation sequence for $t$.

**Example 3.3**

In the computation sequence $cs$ of Example 2.2 we have a superfluous visit to the node $n_3$. We can apply the transformation $T_1$ to $cs$ and get

$$cs_1 = \begin{cases} <n_1, \emptyset> <n_2, 1> \\ <n_5, 1> <n_6, \emptyset> <n_6, S> <n_5, S> <n_2, \emptyset> <n_1, \emptyset> \end{cases}$$

$$cs_3 = \begin{cases} <n_1, \emptyset> <n_2, \emptyset> \\ <n_3, 1> \end{cases}$$

$$cs_4 = \begin{cases} <n_4, \emptyset> <n_4, S> \\ <n_4, \emptyset> <n_4, S> <n_3, S> \\ <n_5, \emptyset> <n_6, \emptyset> <n_6, S> <n_5, S> <n_4, S> <n_5, S> <n_1, S> \end{cases}$$

The transformation $T_2$ can be applied when the computation sequence $cs$ has the form

$$cs = cs_1 <n, B> cs_2 <n, \emptyset> cs_3 <n, \emptyset> cs_4$$

where $|cs_4(n)|$ is even and $cs_2(n) = cs_3(n) = \lambda$. 
Thus we have a visit to $n$ where no attributes become evaluated but there has been at least one more visit to $n$. The result of applying the transformation $T_2$ to $cs$ is the sequence

$$cs' = cs_1 \text{ } cs_3 \text{ } <n, B> \text{ } cs_2 \text{ } cs_4$$

It can again be proved that $cs'$ will be a computation sequence for $t$.

Using the transformations $T_1$ and $T_2$ repeatedly we get the following lemma.

**Lemma 3.4**

For each computation sequence for a derivation tree $t$ there exists a reduced computation sequence for the tree.

**Theorem 3.5**

For any well-defined AG $G$ there exists an integer $k$ such that $G$ is $k$-visit.

**Proof**

Let $K (\geq 1)$ be the maximal number of attributes associated with any non-terminal of $G$. Let $cs$ be a reduced computation sequence for a derivation tree $t$. From the definition of a reduced computation sequence it follows that $|cs(n)| \leq 2K$ for any node $n$ of $t$. Thus $G$ must be $K$-visit.
4. DECIDABILITY OF THE $k$-VISIT PROPERTY

Let $G$ be a well-defined AG and let $k$ be an integer. Is $G$ $k$-visit?
From Theorem 3.5 it follows that there is a $K$ such that $G$ is $K$-visit.
If therefore $k \geq K$ then obviously $G$ is $k$-visit. We will in this section
see how to test whether $G$ is $k$-visit for arbitrary $k > 0$ and thereby
how to find the least $k$ such that $G$ is $k$-visit.

The following lemma is the key for proving the decidability of the $k$-visit
property.

Lemma 4.1
For any well-defined AG $G$ there exists a (computable) integer $q$ such that for all
derivation trees $t$ of $G_u$ there exists a derivation tree $\tilde{t}$ of height at most $q$
such that $t$ is $k$-visit if $\tilde{t}$ is $k$-visit.

Before proving the lemma we need some technical definitions.

Definition 4.2
Let $t$ be a derivation tree and $n$ a node in $t$. An outside($n$) computation
sequence for $t$ is a walk

$$cs = <n_1, A_1> <n_2, A_2> \ldots <n_r, A_r>$$

through $t^n$ satisfying

4) $n_1 = n_r$ is the root of $t$.

5) If $n_j \neq n$ or $A_j \in I(n)$ then the attributes of $A_j$ do not depend on
those in $A_i$ for $i \geq j$, $1 \leq j \leq r$.

6) For each node $n^l$ in $t^n$ (6a)-(6e) are satisfied.
An **inside**(n) computation sequence for $t$ is a walk

$$cs = \langle n_1, A_1 \rangle \langle n_2, A_2 \rangle \ldots \langle n_r, A_r \rangle$$

through $t_n$ satisfying

4$$) \quad n_1 = n_r = n.$$

5$$) \quad \text{If } n_j \neq n \text{ or } A_j \subseteq S(n) \text{ then the attributes of } A_j \text{ do not depend on those of } A_i \text{ for } i \geq j, \ 1 \leq j \leq r.$$

6$$) \quad \text{For each node } n_i \text{ in } t_n \text{ 6a)-6e) are satisfied.}$$

Note that any pair of an **outside**(n) and an **inside**(n) computation sequence $cs_o$ and $cs_i$, for $t$, where $n$ is an interior node, and $cs_o(n) = cs_i(n)$ can be combined into a computation sequence for $t$ and any computation sequence is such a combination.

The concept of a reduced computation sequence can easily be generalized to **outside**(n) and **inside**(n) computation sequences.

**Example 4.3**

For the derivation tree $t$ of Example 2.2 we have an **outside**(n$_3$) computation sequence

$$\langle n_1, \emptyset \rangle \langle n_2, I \rangle \langle n_3, \emptyset \rangle \langle n_3, S \rangle \langle n_5, I \rangle \langle n_6, \emptyset \rangle \langle n_6, S \rangle$$

$$\langle n_5, S \rangle \langle n_3, I \rangle \langle n_3, \emptyset \rangle \langle n_2, S \rangle \langle n_1, S \rangle$$

And we have an **inside**(n$_3$) computation sequence

$$\langle n_3, I \rangle \langle n_4, \emptyset \rangle \langle n_4, S \rangle \langle n_3, \emptyset \rangle \langle n_3, \emptyset \rangle \langle n_3, S \rangle$$
Let \( t \) be a derivation tree and \( n \) a node in \( t \). We define for a reduced inside(\( n \)) computation sequence \( cs \) for \( t \)

\[
\text{visit}^{(i)}(n, cs) = \langle m, cs(n) \rangle
\]

where \( m = \max \left\{ \frac{1}{2} |cs(n')| : n' \text{ in } t^n \right\} \).

We define for a reduced outside(\( n \)) computation sequence \( cs \) for \( t \)

\[
\text{visit}^{(o)}(n, cs) = \langle m, cs(n) \rangle
\]

where \( m = \max \left\{ \frac{1}{2} |cs(n')| : n' \text{ in } t^n \right\} \).

Furthermore, let

\[
\text{VISIT}^{(i)}_t(n) = \{ \text{visit}^{(i)}(n, cs) : cs \text{ is a reduced inside(} n \text{)} \\
\hspace{1cm} \text{computation sequence for } t \}
\]

\[
\text{VISIT}^{(o)}_t(n) = \{ \text{visit}^{(o)}(n, cs) : cs \text{ is a reduced outside(} n \text{)} \\
\hspace{1cm} \text{computation sequence for } t \}
\]

We know that the AG \( G \) is \( K \)-visit where \( K \) is the maximal number of attributes associated with any nonterminal of \( G \). Therefore in any tuple \( \langle m, B, 1, B_2, \ldots, B_h \rangle \) in \( \text{VISIT}^{(i)}_t(n) \) (resp. \( \text{VISIT}^{(o)}_t(n) \)) we have \( m \leq K \) and \( h \leq 2m \). This means that the number of elements in \( \text{VISIT}^{(i)}_t(n) \) (resp. \( \text{VISIT}^{(o)}_t(n) \)) is bounded by some (computable) constant \( V_G \), which depends only on the AG \( G \).

Consider a derivation tree \( t \) of \( G_u \). Assume that in \( t \) we have two nodes \( n \) and \( n' \) satisfying

i) \( n' \) is a descendant of \( n \)

ii) \( n \) and \( n' \) have the same nonterminal \( X \) as label

iii) \( \text{VISIT}^{(i)}_t(n) = \text{VISIT}^{(i)}_t(n') \).
Let $t'$ be the derivation tree obtained by replacing (in $t$) the subtree $t_n$ with the subtree $t_{n'}$.

We will prove that if $t'$ is $k$-visit then $t$ is $k$-visit. Assume that $t'$ is $k$-visit. Then there exist a reduced computation sequence $cs_t'$ for $t'$ and elements $<m_1, cs_t(n')>$ in $VISIT^{(i)}_{t'}(n')$ and $<m_2, cs_t(n')>$ in $VISIT^{(o)}_{t'}(n')$ such that $k \geq \max \{ m_1, m_2 \}$. Since $VISIT^{(i)}_t(n) = VISIT^{(i)}_{t'}(n') = VISIT^{(i)}_{t'}(n')$ and $VISIT^{(o)}_t(n) = VISIT^{(o)}_{t'}(n')$ there exist an inside$(n)$ computation sequence $cs_i$ for $t$, such that $cs_i(n) = cs_t(n')$ and $<m_1, cs_i(n)>$ is in $VISIT^{(i)}_t(n)$, and an outside$(n)$ computation sequence $cs_o$ for $t$ such that $cs_o(n) = cs_t(n')$ and $<m_2, cs_o(n)>$ is in $VISIT^{(o)}_t(n)$. $t$ is then $k$-visit because $cs_i$ and $cs_o$ can be combined to a computation sequence, which visits each node at most $\max \{ m_1, m_2 \} \leq k$ times.

To complete the proof of the Lemma observe that if the height of $t$ is greater than $q = N_G \cdot 2^{|V_G|} + 1$ where $N_G$ is the number of nonterminals in $G$, then there exist nodes $n$ and $n'$ in $t$ satisfying i)-iii) above. By removing parts of the tree as above repeatedly we can finally obtain a derivation tree $\tilde{t}$ with the required property.

$\square$
**Theorem 4.4**
For any well-defined AG $G$ and any integer $k$ it is decidable whether $G$ is $k$-visit or not.

**Proof**
Lemma 4.1 shows that we only have to check whether all derivation trees of height bounded by some (computable) constant are $k$-visit.

**Corollary 4.5**
Given a well-defined AG $G$ it is possible to find the minimal $k$ for which $G$ is $k$-visit.
5. A k-visit Hierarchy

In this section we will give an example (from [8]) of a translation that can be specified by a k-visit AG but which cannot be specified by any (k-1)-visit AG over the same semantic domain.

The AG will be over a semantic domain $S = (D, F)$. $D$ consists of a single domain $D = A^*$, i.e. sequences of $A$'s. In $F$ we have two semantic functions

$$1_\lambda : \rightarrow D \quad 1_\lambda(\lambda) = \lambda, \quad \text{the empty string}$$

$$\varepsilon_A : D \rightarrow D \quad \varepsilon_A(a) = Aa, \quad \text{concatenation with } A, \ a \in D$$

First we specify the productions of the underlying grammar $G_u$:

$$Z ::= X$$

$$X ::= A X | X$$

We will define an AG $G_k$ with underlying grammar $G_u$ which specifies the translation

$$\tau(k) = \{ (t, v^{2k}) : t \text{ is a derivation tree for } v \text{ in } G_u \}.$$  

The nonterminal $X$ has associated $2 \cdot k$ attributes, $k$ inherited and $k$ synthesized and $Z$ has one synthesized attribute. Let $G_k$ have the productions:

p1: $<Z \downarrow \varepsilon_A(a_k)> ::=$

$$<X \downarrow 1_\lambda(\lambda) \downarrow \varepsilon_A(a_1) \downarrow \varepsilon_A(a_2) \ldots \downarrow \varepsilon_A(a_{k-1}) \uparrow a_1 \uparrow a_2 \ldots \uparrow a_k>$$

p2: $<X \downarrow a_1 \downarrow a_2 \ldots \downarrow a_k \uparrow \varepsilon_A(b_1) \uparrow \varepsilon_A(b_2) \ldots \uparrow \varepsilon_A(b_k)> ::=$

$$A <X \downarrow \varepsilon_A(a_1) \downarrow \varepsilon_A(a_2) \ldots \downarrow \varepsilon_A(a_k) \uparrow b_1 \uparrow b_2 \ldots \uparrow b_k>$$

p3: $<X \downarrow a_1 \downarrow a_2 \ldots \downarrow a_k \uparrow \varepsilon_A(a_1) \uparrow \varepsilon_A(a_2) \ldots \uparrow \varepsilon_A(a_k)> ::= A$

It is easy to see that $T(G_k, \emptyset) = \tau(k)$.  

As an example consider the following derivation tree \((k = 2)\) and its "dependency graph".

A (reduced) computation sequence for this tree is

\[
\langle n_1, \emptyset \rangle \langle n_2, l_1 \rangle \langle n_3, l_1 \rangle \langle n_4, l_1 \rangle \langle n_4, S_1 \rangle \langle n_3, S_1 \rangle \langle n_3, S_1 \rangle \\
\langle n_2, l_2 \rangle \langle n_3, l_2 \rangle \langle n_4, l_2 \rangle \langle n_4, S_2 \rangle \langle n_3, S_2 \rangle \langle n_3, S_2 \rangle \\
\langle n_1, S_1 \rangle
\]

where \(l_j\) is the \(j\)th inherited attribute of a node and \(S_j\) is the \(j\)th synthesized, \(1 \leq j \leq k\).

In general consider a derivation tree \(t\) of \(G_k\).
The following will be a reduced computation sequence for $t$:

$$cs = \langle n_1, \emptyset \rangle \: cs_1 \: cs_2 \ldots \: cs_k \: \langle n_1, S_1 \rangle$$

where

$$cs_j = \langle n_2^j, l_j \rangle \: \langle n_3^j, l_j \rangle \ldots \: \langle n_m^j, l_j \rangle$$

$$\langle n_m^j, S_j \rangle \: \langle n_{(m-1)}^j, S_j \rangle \ldots \: \langle n_2^j, S_j \rangle.$$  

Since $|cs(n)| = 2k$ for $n \neq n_1$ and $|cs(n_1)| = 2$ it is obvious that $G_k$ is $k$-visit.

We claim that the translation $\tau(k)$ cannot be specified by a $(k-1)$-visit AG over the semantic domain $\emptyset$.

To see this, assume that $G_k$ is a $(k-1)$-visit AG over the semantic domain $\emptyset$ such that $T(G_k, \emptyset) = \tau(k)$ and let $t$ be the derivation tree for $A^m$ as above. The length of any reduced computation sequence for $t$ in $G_k$ will be less than or equal to $2(k-1) \cdot m+2$ since each internal node is visited at most $k-1$ times and the root of $t$ only once. Each application of a semantic function from $F$ can at most increase the length of the value of the "new" attribute by one compared with the "old" one. Thus the length of $\text{meaning}(t, \emptyset)$ is at most $2(k-1) \cdot m+1$ (we must start by applying $1(\lambda)$ which contradicts the fact that the length equals $2k \cdot m$.

Altogether this proves the following theorem.

**Theorem 5.1**

There exists a semantic domain $\emptyset$ such that translations specified by $k$-visit AGs over $\emptyset$ define a proper hierarchy.
References

[1] G. V. Bochmann:

"The formal power of one-visit attribute grammars"
Memorandum 286, Twente University of Technology, 1979.

"The intrinsically exponential complexity of the circularity problem

"The alternating semantic evaluator"

[5] M. Kastens:
"Ordered attribute grammars", Acta Informatica, vol. 13,
Fasc. 3, 229-256, 1980.

"Automatic generation of efficient evaluators for attribute grammars",
Conference Record of the Third ACM Symposium on Principles
of Programming Languages, 32-49, 1976.

[7] D. E. Knuth:
"Semantics of context-free languages", Math. Syst. Theory 2,

[8] B. H. Mayoh:
"Attribute grammars and mathematical semantics"

[9] H. Riis:
"Subclasses of attribute grammars"
DAIMI PB-114, Dept. of Comp. Sci., Aarhus University, 1980.