Attribute Grammars


Reinhard Wilhelm
Universität des Saarlandes
wilhelm@cs.uni-saarland.de

5. November 2013
Attribute Grammars

Attributes: containers for static semantic (non-context-free syntactic) information,

Directions: attributes

  inherit information from the (upper) context,
  synthesize information from information in subtrees,

Semantic rules: define computation of attribute values.
Attributes as Carriers of Context Information

Inherited

Synthesized
Example Grammar: Scoping

Describes nested scopes;

- a statement may be a block, consisting of a declaration part followed by a statement part,
- declaration parts consist of lists of procedure declarations,
- procedures, declared later in a list, may be called from within procedures declared earlier.

attribute grammar Scopes:

donterminals Stms, Stm, Decls, Decl, Id, Args, Ptype;

domain Env = String → Types;

attributes syn ok with Decls, Decl, Stms, Stm domain Bool;
inh e-env with Stms, Stm, Decls, Decl domain Env;
inh it-env with Decls, Decl domain Env;
syn st-env with Decls, Decl domain Env;
syn name with Id domain String;
syn type with Ptype, Args domain Types;
ok is true,
  ▶ if all used identifiers are declared, and
  ▶ if there are no multiple declarations of one identifier in the same scope.

\textit{it-env, st-env} are “temporary environments”, in which declarative information is collected.
A check for double declarations is made while collecting local declarations in \textit{it-env}.

\textit{e-env} is the “effective” environment, in which procedure calls are type checked.
For each nested scope, the effective environment is obtained by over-writing the external effective environment with the locally constructed environment.
rules
0: \( Stms \rightarrow Stm \)
1: \( Stms \rightarrow Stms ; Stm \)
   \( Stms_0.ok = Stms_1.ok \) and \( Stm.ok \)
2: \( Stm \rightarrow \textbf{begin} \ Decls \; \textbf{end} \)
   \( Decls.it-env = \emptyset \)
   \( Stms.e-env = Stm.e-env + Decls.st-env \)
   \( Decls.e-env = Stm.e-env + Decls.st-env \)
   \( Stm.ok = Decls.ok \) and \( Stms.ok \)
3: \( Decls \rightarrow Decl \)
4: \( Decls \rightarrow Decls ; Decl \)
   \( Decls_1.it-env = Decls_0.it-env \)
   \( Decl.it-env = Decls_1.st-env \)
   \( Decls_0.st-env = Decl.st-env \)
   \( Decls_0.ok = Decls_1.ok \) and \( Decl.ok \)
5: \( Decl \rightarrow \textbf{proc} \ Id : Ptype \; \textbf{is} \ Stms \)
   \( Decl.st-env = Decl.it-env + \{ \text{Id.name} \mapsto \text{Ptype.type} \} \)
   \( Stms.e-env = Decl.e-env \)
   \( Decl.ok = \text{undef}(\text{Id.name}, Decl.it-env) \) and \( Stms.ok \)
6: \( Stm \rightarrow \textbf{call} \ Id \ ( \text{Args} ) \)
   \( Stm.ok = \text{def}(\text{Id.name}, Stm.e-env) \) and
   \( \text{check}(\text{Args.type}, Stm.e-env(\text{Id.name})) \)
Local Dependencies in the Scopes-AG

1. e-env ok Stms
   Stms
   Stms

2. e-env ok Stm
   Stm
   Stms
   Decls
   it-env
   st-env

4. e-env it-env st-env ok
   Decls
   Decls
   Decls
   Decls

5. e-env it-env st-env ok
   Decl
   Decl
   Decl
   Decl
   Stms
   Id
   Ptype
   Stms

6. e-env ok Stm
   Stm
   Id
   Args
Attribute Grammars – Terminology

Let $G = (V_N, V_T, P, S)$ be a CFG, the underlying CFG. The $p$–th production in $P$ is written as $p : X_0 \rightarrow X_1 \ldots X_{n_p}$, $X_i \in V_N \cup V_T$, $1 \leq i \leq n_p$, $X_0 \in V_N$.

An attribute grammar (AG) over $G$ consists of

- two disjoint sets $Inh$ and $Syn$ of inherited resp. synthesized attributes,
- an association of two sets $Inh(X) \subseteq Inh$ and $Syn(X) \subseteq Syn$ with each symbol in $V_N \cup V_T$;
  - $Attr(X) = Inh(X) \cup Syn(X)$ set of all attributes of $X$;
  - $a \in Attr(X_i)$ has an occurrence in production $p$ at occurrence $X_i$, written $a_i$.
  - $O(p)$ is the set of all attribute occurrences in production $p$. 
Attribute Grammars – Terminology cont’d

- the association of a **domain** $D_a$ with each attribute $a$;
- a **semantic rule**

$$a_i = f_{p,a,i}(b_{j_1}^1, \ldots, b_{j_k}^k) \quad (0 \leq j_l \leq n_p) \ (1 \leq l \leq k)$$

for each **defining occurrence** of an attribute, i.e.,

- $a \in Inh(X_i)$ for $1 \leq i \leq n_p$ or
- $a \in Syn(X_0)$ in each production $p$,

where $b_{j_l}^l \in Attr(X_{j_l}) \ (0 \leq j_l \leq n_p) \ (1 \leq l \leq k)$. $f_{p,a,i}$ is thus a function from $D_{b_1^1} \times \ldots \times D_{b_k^k}$ to $D_a$. 


Attributes as Carriers of Context Information

Inherited

Synthesized
More Terminology

- Productions of the *underlying* CFG have **instances** in syntax trees.

- Node $n$ labelled with $X \in V_N \cup V_T$ has an **instance** $a_n$ of attribute $a \in Attr(X)$.

- Hence, there are

  - attributes associated with non-terminals (and terminals),
  - attribute occurrences in productions, and
  - attribute instances at nodes of syntax trees.

- The semantic rule for a def. attribute occurrence in a production determines the values of all corresponding attribute instances in instances of the production.

- **Attribute Evaluation** is the process of computing the values of attribute instances in a tree using the semantic rules.
Attribute Occurrences and Attribute Instances

A production and one of its instances

attribute occurrences

\[ a_0, a_1 \]

attribute instances

\[ a_n, a_{n1} \]
The p–n–q Situation

Attribute evaluation at node \( n \) labelled \( X \) is determined by productions

\[
p \quad \text{applied at } parent(n) \text{ for the inherited attributes of } X \quad \text{and} \quad q \quad \text{applied at } n \text{ for the synthesized attributes of } X.
\]
Semantics of an Attribute Grammar

Let $t$ be a syntax tree to AG $G$, $\text{symb}(n) \in V_N$, $\text{prod}(n)$ be the production applied at $n$.
Attribute instance $a_n$ of attribute $a \in \text{Attr}(\text{symb}(n))$ at $n$ has to be given a value from $D_a$.
Semantic rule $a_i = f_{p,a,i}(b_{j_1}^1, \ldots, b_{j_k}^k)$ of $\text{prod}(n) = p$ induces the relation on the values of the attribute instances of the instance of $\text{prod}(n)$:

$$\text{val}(a_{ni}) = f_{p,a,i}(\text{val}(b_{nj_1}^1), \ldots, \text{val}(b_{nj_k}^k))$$

$G$ induces a system of equations for $t$:

- variables are the attribute instances at the nodes of $t$,
- equations are defined by the above relation,
- recursion would in general not permit an evaluation of all attribute instances.
- AG, which never induces a recursive system of equations, is called well formed.
Normal Form

- Attribute occurrences $a_i$ where $a \in Inh(X_i)$ and $1 \leq i \leq n_p$ or $a \in Syn(X_0)$ are **defining occurrences**.
- All others are **applied occurrences**.
- AG is in **normal form**, if all arguments of semantic functions are applied occurrences.

Consequences of Normal Form:

- Semantic rules define values of def. occurrences in terms of appl. occurrences.
- Computation of the value of an attribute in one instance of a production (in a tree) requires the previous evaluation of an attribute in a neighbouring instance of a production.
- For later: Chains of attribute dependences inside a production have at most length one.
Short Circuit Evaluation of Boolean Expressions

The generated code:

- only load-instructions and conditional jumps;
- no instructions for **and**, **or** and **not**;
- subexpressions evaluated from left to right;
- for each (sub)expression, only the smallest subexpression is evaluated, which determines the value of the whole (sub)expression.
Code for the Boolean expression \((a \text{ and } b) \text{ or not } c\):

\begin{align*}
\text{LOAD } a \\
\text{JUMPF } L1 & \quad \text{jump-on-false} \\
\text{LOAD } b \\
\text{JUMPT } L2 & \quad \text{jump-on-true} \\
L1: & \quad \text{LOAD } c \\
\text{JUMPT } L3 \\
L2: & \quad \text{Code for true–successor} \\
L3: & \quad \text{Code for false–successor}
\end{align*}
Attribute grammar **BoolExp** describes

- code generation for short circuit evaluation,
- label generation for subexpressions,
- transport of labels for true– and false–successors to primitive subexpressions translated into jumps.
Synthesized attribute \( j\text{cond} \) computes the correlation of the values of an expression with that of its rightmost identifier \( x \).

Value of \( j\text{cond} \) at expression \( e \)

- \textit{true}: The loaded value of \( x \) equals value of \( e \),
- \textit{false}: The loaded value of \( x \) is negation of value of \( e \).

Means for code generation:
Instruction following \texttt{LOAD} \( x \) is conditional jump to true–successor

- \texttt{JUMPT} if \( j\text{cond} = \text{true} \),
- \texttt{JUMPF} if \( j\text{cond} = \text{false} \).
attribute grammar BoolExp

nonterminals IFSSTAT, STATS, E, T, F;

attributes inh tsucc, fsucc with E,T,F domain string;
syn jcond with E,T,F domain bool;
syn code with IFSSTAT, E,T,F domain string;
rules
IFSTAT \rightarrow \textbf{if} \ E \ \textbf{then} \ STATS \ \textbf{else} \ STATS \ \textbf{fi}
\[
\begin{align*}
E.\text{tsucc} &= \ t \\
E.\text{fsucc} &= \ e \\
\text{IFSTAT}.\text{code} &= \ E.\text{code} ++ \ \text{gencjump} \ (\textbf{not} \ E.\text{jcond}, \ e) ++ \\
t: ++ \ \text{STATS}_1.\text{code} ++ \ \text{genujump} \ (f) ++ \ e: ++ \ \text{STATS}_2.\text{code} ++ \ f:
\end{align*}
\]
E \rightarrow T
E \rightarrow E \ \textbf{or} \ T
\[
\begin{align*}
E_1.\text{fsucc} &= \ t \\
E_0.\text{jcond} &= \ T.\text{jcond} \\
E_0.\text{code} &= \ E_1.\text{code} ++ \ \text{gencjump} \ (E_1.\text{jcond}, \ E_0.\text{tsucc}) ++ \ t: ++ \ T.\text{code}
\end{align*}
\]
T \rightarrow F
T \rightarrow T \ \textbf{and} \ F
\[
\begin{align*}
T_1.\text{tsucc} &= \ f \\
T_0.\text{jcond} &= \ F.\text{jcond} \\
T_0.\text{code} &= \ T_1.\text{code} ++ \ \text{gencjump} \ (\textbf{not} \ T_1.\text{jcond}, \ T_0.\text{fsucc}) ++ \ f: ++ \ F.\text{code}
\end{align*}
\]
F \rightarrow (E)
F \rightarrow \textbf{not} \ F
\[
\begin{align*}
F_1.\text{tsucc} &= \ F_0.\text{fsucc} \\
F_1.\text{fsucc} &= \ F_0.\text{tsucc} \\
F_0.\text{jcond} &= \ \textbf{not} \ F_1.\text{jcond}
\end{align*}
\]
F \rightarrow \textbf{id}
\[
\begin{align*}
F.\text{jcond} &= \ true \\
F.\text{code} &= \ \text{LOAD} \ \textbf{id}.\text{identifier}
\end{align*}
\]
Auxiliary functions:

genujump (l) = JUMP l

genjump (jc, l) = if jc = true
  then JUMPT l
  else JUMPF l
  fi