



Compiler Construction

Lecture 12: Semantic Analysis II (Circularity of Attribute Grammars)

Summer Semester 2017

Thomas Noll

Software Modeling and Verification Group

RWTH Aachen University

<https://moves.rwth-aachen.de/teaching/ss-17/cc/>

Recap: Attribute Grammars

Attribute Grammars I

Goal: compute context-dependent but runtime-independent properties of a given program

Idea: enrich context-free grammar by **semantic rules** which annotate syntax tree with **attribute values**

⇒ **Semantic analysis = attribute evaluation**

Result: **attributed syntax tree**

In greater detail:

- With every grammar symbol a set of attributes is associated.
- Two types of attributes are distinguished:
 - Synthesized:** bottom-up computation (from the leaves to the root)
 - Inherited:** top-down computation (from the root to the leaves)
- With every production a set of semantic rules is associated.

Recap: Attribute Grammars

Example: Type Checking I

Example (Attribute grammar for type checking)

$Pgm \rightarrow Dcl\ Cmd$	$ok.0 = ok.2$	$env.2 = st.1$
$Dcl \rightarrow \varepsilon$	$st.0 = [id \mapsto err \mid id \in Id]$	
$Typ\ var ; Dcl$	$st.0 = st.4[id.2 \mapsto typ.1]$	
$Typ \rightarrow int$	$typ.0 = int$	
$Typ \rightarrow bool$	$typ.0 = bool$	
$Cmd \rightarrow \varepsilon$	$ok.0 = true$	
$var := Exp ; Cmd$	$ok.0 = (env.0(id.1) = typ.3 \wedge ok.5)$	$env.3 = env.0\ env.5 = env.0$
$Exp \rightarrow num$	$typ.0 = int$	
var	$typ.0 = env.0(id.1)$	
$Exp + Exp$	$typ.0 = (typ.1 = typ.3 = int ? int : err)$	$env.1 = env.0\ env.3 = env.0$
$Exp < Exp$	$typ.0 = (typ.1 = typ.3 = int ? bool : err)$	$env.1 = env.0\ env.3 = env.0$
$Exp \&\& Exp$	$typ.0 = (typ.1 = typ.3 = bool ? bool : err)$	$env.1 = env.0\ env.3 = env.0$

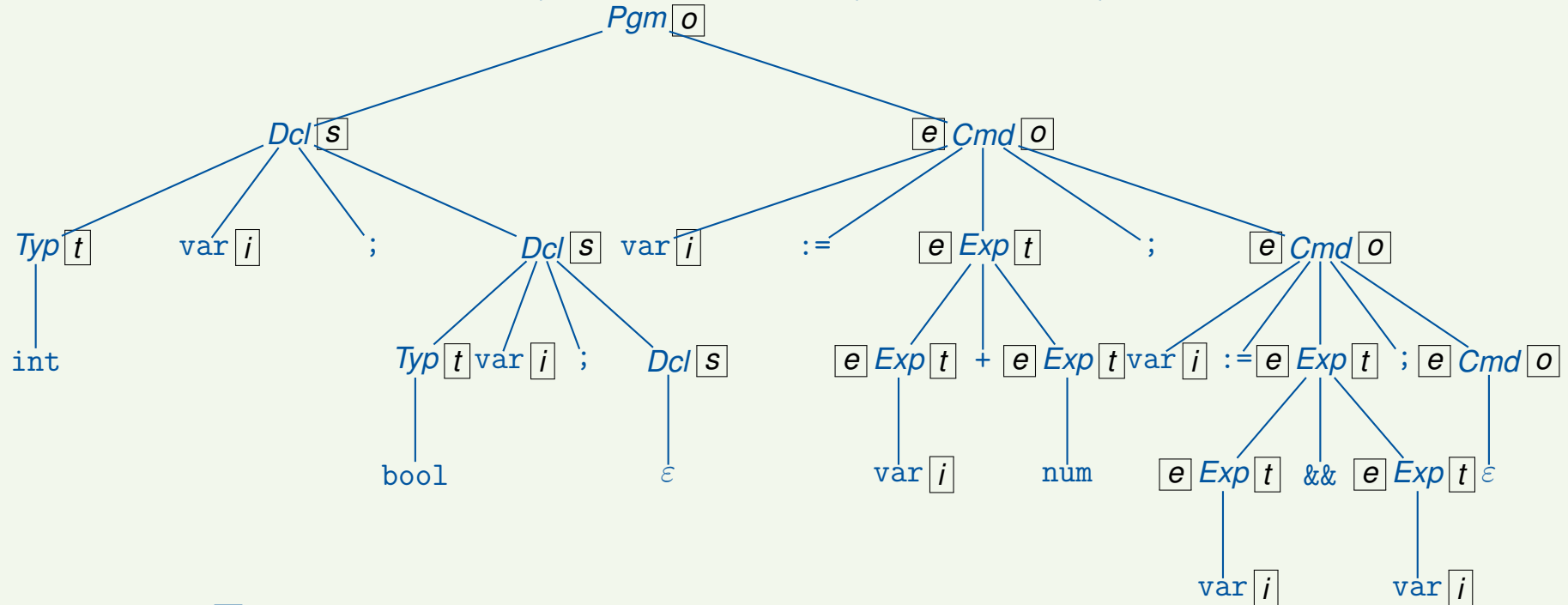
- **Synthesized attributes:** id (identifier name), ok (Boolean result), st (symbol table, mapping identifiers to types), typ (data type in $\{bool, int, err\}$)
- **Inherited attributes:** env (environment – same type as symbol table)

Recap: Attribute Grammars

Example: Type Checking II

Example (Attributed syntax tree)

For program `int x; bool y; x := x+1; y := x && y:`



($[e] = env, [i] = id, [o] = ok, [s] = st, [t] = typ$)

Formal Definition of Attribute Grammars

Formal Definition of Attribute Grammars I

Definition 12.1 (Attribute grammar)

Let $G = \langle N, \Sigma, P, S \rangle \in CFG_{\Sigma}$ with $X := N \uplus \Sigma$.

- Let $Att = Syn \uplus Inh$ be a set of (synthesized or inherited) attributes, and let $V = \bigcup_{\alpha \in Att} V^{\alpha}$ be a union of value sets.
- Let $att : X \rightarrow 2^{Att}$ be an attribute assignment, and let $\text{syn}(Y) := att(Y) \cap Syn$ and $\text{inh}(Y) := att(Y) \cap Inh$ for every $Y \in X$.

- Every production $\pi = Y_0 \rightarrow Y_1 \dots Y_r \in P$ determines the set

$$Var_{\pi} := \{\alpha.i \mid \alpha \in att(Y_i), i \in \{0, \dots, r\}\}$$

of attribute variables of π with the subsets of inner and outer variables:

$$In_{\pi} := \{\alpha.i \mid (i = 0, \alpha \in \text{syn}(Y_i)) \text{ or } (i \in [r], \alpha \in \text{inh}(Y_i))\} \quad Out_{\pi} := Var_{\pi} \setminus In_{\pi}$$

- A semantic rule of π is an equation of the form

$$\alpha_0.i_0 = f(\alpha_1.i_1, \dots, \alpha_n.i_n)$$

where $n \in \mathbb{N}$, $\alpha_0.i_0 \in In_{\pi}$, $\alpha_j.i_j \in Out_{\pi}$, and $f : V^{\alpha_1} \times \dots \times V^{\alpha_n} \rightarrow V^{\alpha_0}$.

- For each $\pi \in P$, let E_{π} be a set with exactly one semantic rule for every inner variable of π , and let $E := (E_{\pi} \mid \pi \in P)$.

Then $\mathfrak{A} := \langle G, E, V \rangle$ is called an attribute grammar: $\mathfrak{A} \in AG$.

Formal Definition of Attribute Grammars

Formal Definition of Attribute Grammars II

Example 12.2 (cf. Example 11.1)

$\mathcal{A} \in AG$ for type checking:

- **Attributes:** $Att = Syn \uplus Inh$ with $Syn = \{id, ok, st, typ\}$ and $Inh = \{env\}$
- **Value sets:** $V^{id} \subseteq \Omega^+$, $V^{ok} = \mathbb{B}$, $V^{st} = V^{id} \rightarrow V^{typ}$, $V^{typ} = \{\text{bool, int, err}\}$
- **Attribute assignment:**

$Y \in X$	Pgm	Dcl	Typ	Cmd	Exp	var	$a \in \Sigma \setminus \{\text{var}\}$
$\text{syn}(Y)$	$\{ok\}$	$\{st\}$	$\{typ\}$	$\{ok\}$	$\{typ\}$	$\{id\}$	\emptyset
$\text{inh}(Y)$	\emptyset	\emptyset	\emptyset	$\{env\}$	$\{env\}$	\emptyset	\emptyset

- **Attribute variables:**

$\pi \in P$	$Pgm \rightarrow Dcl \text{ } Cmd$	$Cmd \rightarrow \text{var} := Exp; Cmd$	$Exp \rightarrow Exp + Exp$...
In_π	$\{ok.0, env.2\}$	$\{ok.0, env.3, env.5\}$	$\{typ.0, env.1, env.3\}$...
Out_π	$\{st.1, ok.2\}$	$\{env.0, id.1, typ.3, ok.5\}$	$\{env.0, typ.1, typ.3\}$...

- **Semantic rules:** see Example 11.1 (e.g., $E_{Pgm \rightarrow Dcl \text{ } Cmd} = \{ok.0 = ok.2, env.2 = st.1\}$)

The Attribute Equation System

Attribution of Syntax Trees I

Definition 12.3 (Attribution of syntax trees)

Let $\mathcal{A} = \langle G, E, V \rangle \in AG$, and let t be a syntax tree of G with the set of nodes K .

- K determines the set of **attribute variables of t** :

$$Var_t := \{\alpha.k \mid k \in K \text{ labelled with } Y \in X, \alpha \in \text{att}(Y)\}.$$

- Let $k_0 \in K$ be an (inner) node where production $\pi = Y_0 \rightarrow Y_1 \dots Y_r \in P$ is applied, and let $k_1, \dots, k_r \in K$ be the corresponding successor nodes. The **attribute equation system E_{k_0}** of k_0 is obtained from E_π by substituting every attribute index $i \in \{0, \dots, r\}$ by k_i .
- The **attribute equation system** of t is given by

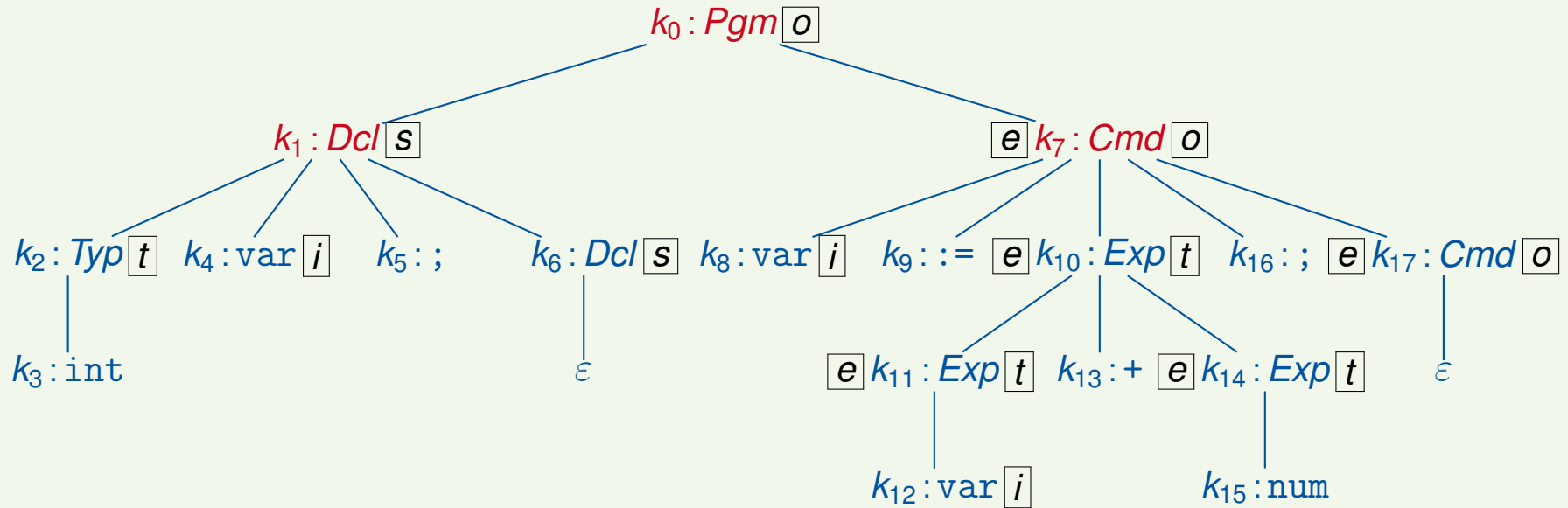
$$E_t := \bigcup \{E_k \mid k \text{ inner node of } t\}.$$

The Attribute Equation System

Attribution of Syntax Trees II

Example 12.4 (cf. Example 11.1)

Attributed syntax tree for `int x; x := x+1;;`



$$E_{Pgm \rightarrow Dcl \text{ } Cmd} : \begin{array}{l} ok.0 = ok.2 \\ env.2 = st.1 \end{array} \xrightarrow{\text{subst}} E_{k_0} : \begin{array}{l} ok.k_0 = ok.k_7 \\ env.k_7 = st.k_1 \end{array}$$

The Attribute Equation System

Attribution of Syntax Trees III

Corollary 12.5

For each $\alpha.k \in \text{Var}_t$ except for the inherited attribute variables at the root and the synthesized attribute variables at the leaves of t , E_t contains *exactly one equation* with left-hand side $\alpha.k$.

Assumptions:

- The **start symbol** does not have inherited attributes: $\text{inh}(S) = \emptyset$.
- **Synthesized attributes of terminal symbols** are provided by the scanner.

Circularity of Attribute Grammars

Solvability of Attribute Equation System I

Definition 12.6 (Solution of attribute equation system)

Let $\mathfrak{A} = \langle G, E, V \rangle \in AG$, and let t be a syntax tree of G . A **solution** of E_t is a mapping

$$v : Var_t \rightarrow V$$

such that, for every $\alpha_0.k_0 \in Var_t$ and $\alpha_0.k_0 = f(\alpha_1.k_1, \dots, \alpha_n.k_n) \in E_t$,

$$v(\alpha_0.k_0) = f(v(\alpha_1.k_1), \dots, v(\alpha_n.k_n)).$$

In general, the attribute equation system E_t of a given syntax tree t can have

- no solution,
- exactly one solution, or
- several solutions.

Circularity of Attribute Grammars

Solvability of Attribute Equation System II

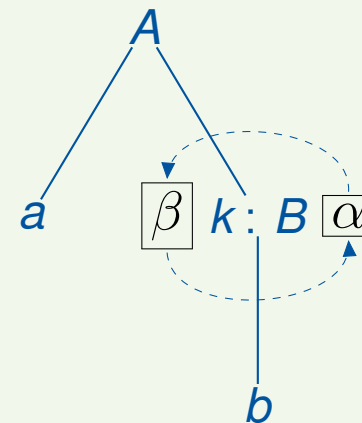
Example 12.7

- $A \rightarrow aB, B \rightarrow b \in P$
- $\alpha \in \text{syn}(B), \beta \in \text{inh}(B)$
- $\beta.2 = f(\alpha.2) \in E_{A \rightarrow aB}$
- $\alpha.0 = \beta.0 \in E_{B \rightarrow b}$

\implies for $V^\alpha := V^\beta := \mathbb{N}$ and

- $f(x) := x + 1$: **no solution**
- $f(x) := 2x$: **exactly one solution**
($v(\alpha.k) = v(\beta.k) = 0$)
- $f(x) := x$: **infinitely many solutions**
($v(\alpha.k) = v(\beta.k) = y$ for any $y \in \mathbb{N}$)

\implies **cyclic dependency:**



$$E_t : \begin{aligned} \beta.k &= f(\alpha.k) \\ \alpha.k &= \beta.k \end{aligned}$$

Circularity of Attribute Grammars

Circularity of Attribute Grammars

Goal: **unique solvability** of equation system

⇒ avoid cyclic dependencies

Definition 12.8 (Circularity)

An attribute grammar $\mathcal{A} = \langle G, E, V \rangle \in AG$ is called **circular** if there exists a syntax tree t such that the attribute equation system E_t is recursive (i.e., some attribute variable of t depends on itself). Otherwise it is called **noncircular**.

Remark: because of the division of Var_π into In_π and Out_π , cyclic dependencies cannot occur at production level.