Compiler Construction

Lecture 17: Semantic Analysis III (Circularity Test & Attribute Evaluation)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH Aachen University

noll@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/cc10/

Winter semester 2010/11

Outline

Formal Definition of Attribute Grammars

Definition (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 $\text{att} : X \to 2^{Att}$ be an attribute assignment, and let $\text{syn}(Y) := \text{att}(Y) \cap Syn$ and $\text{inh}(Y) := \text{att}(Y) \cap Inh$ for every $Y \in X$.
- Every production $\pi = Y_0 \to Y_1 \dots Y_r \in P$ determines the set
  $$Var_\pi := \{\alpha.i \mid \alpha \in \text{att}(Y_i), i \in \{0, \dots, r\}\}$$
  of attribute variables of π with the subsets of inner and outer variables:
  $$\begin{aligned} In_\pi &:= \{\alpha.i \mid (i = 0, \alpha \in \text{syn}(Y_i)) \text{ or } (i \in [r], \alpha \in \text{inh}(Y_i))\} \\ Out_\pi &:= Var_\pi \setminus In_\pi \end{aligned}$$
- A semantic rule of π is an equation of the form
  $$\alpha.i = f(\alpha_1.i_1, \dots, \alpha_n.i_n)$$
  where $n \in \mathbb{N}$, $\alpha.i \in In_\pi$, $\alpha_j.i_j \in Out_\pi$, and $f : V^{\alpha_1} \times \dots \times V^{\alpha_n} \to V^\alpha$.
- 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: $\mathfrak{A} \in AG$.

Goal: unique solvability of equation system

$\Longrightarrow$ avoid cyclic dependencies

Definition (Circularity)

An attribute grammar $\mathfrak{A} = \langle G, E, V \rangle \in AG$ 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 (see Corollary 16.8).

Attribute Dependency Graphs and Circularity I

Observation: a cycle in the dependency graph D_t of a given syntax tree t is caused by the occurrence of a "cover" production $\pi = A_0 \to w_0 A_1 w_1 \dots A_r w_r \in P$ in a node k_0 of t such that

- the dependencies in E_{k_0} yield the "upper end" of the cycle and
- for at least one $i \in [r]$, some attributes in $\text{syn}(A_i)$ depend on attributes in $\text{inh}(A_i)$.

Example

on the board

To identify such "critical" situations we need to determine for each $i \in [r]$ the possible ways in which attributes in $\text{syn}(A_i)$ can depend on attributes in $\text{inh}(A_i)$.

Attribute Dependency Graphs and Circularity II

Definition (Attribute dependence)

Let $\mathfrak{A} = \langle G, E, V \rangle \in AG$ with $G = \langle N, \Sigma, P, S \rangle$.

- If t is a syntax tree with root label $A \in N$ and root node k, $\alpha \in \text{syn}(A)$, and $\beta \in \text{inh}(A)$ such that $\beta.k \to_t^+ \alpha.k$, then α is dependent on β below A in t (notation: $\beta \overset{A}{\hookrightarrow} \alpha$).
- For every syntax tree t with root label $A \in N$,
$$is(A, t) := \{(\beta, \alpha) \in \text{inh}(A) \times \text{syn}(A) \mid \beta \overset{A}{\hookrightarrow} \alpha \text{ in } t\}.$$
- For every $A \in N$,
$$IS(A) := \{is(A, t) \mid t \text{ syntax tree with root label A}\} \subseteq 2^{Inh \times Syn}.$$

Remark: it is important that $IS(A)$ is a system of attribute dependence sets, not a union (later: strong noncircularity).

Example

on the board

Outline

The Circularity Test I

In the circularity test, the dependency systems $IS(A)$ are iteratively computed. It employs the following notation:

Definition 17.1

Given $\pi = A \to w_0 A_1 w_1 \ldots A_r w_r \in P$ and $is_i \subseteq \mathrm{inh}(A_i) \times \mathrm{syn}(A_i)$ for every $i \in [r]$, let

$$is[\pi; is_1, \ldots, is_r] \subseteq \mathrm{inh}(A) \times \mathrm{syn}(A)$$

be given by

$$is[\pi; is_1, \ldots, is_r] :=$$
$$\Big\{(\beta, \alpha) \mid (\beta.0, \alpha.0) \in (\to_\pi \cup \textstyle\bigcup_{i=1}^r \{(\beta'.p_i, \alpha'.p_i) \mid (\beta', \alpha') \in is_i\})^+\Big\}$$

where $p_i := \sum_{j=1}^i |w_{j-1}| + i$.

In the circularity test, the dependency systems $IS(A)$ are iteratively computed. It employs the following notation:

Definition 17.1

Given $\pi = A \to w_0 A_1 w_1 \ldots A_r w_r \in P$ and $is_i \subseteq \text{inh}(A_i) \times \text{syn}(A_i)$ for every $i \in [r]$, let

$$is[\pi; is_1, \ldots, is_r] \subseteq \text{inh}(A) \times \text{syn}(A)$$

be given by

$$is[\pi; is_1, \ldots, is_r] :=$$
$$\left\{(\beta, \alpha) \mid (\beta.0, \alpha.0) \in \left(\to_\pi \cup \bigcup_{i=1}^{r} \{(\beta'.p_i, \alpha'.p_i) \mid (\beta', \alpha') \in is_i\}\right)^+\right\}$$

where $p_i := \sum_{j=1}^{i} |w_{j-1}| + i$.

Example 17.2

on the board

The Circularity Test II

Algorithm 17.3 (Circularity test for attribute grammars)

Input: $\mathfrak{A} = \langle G, E, V \rangle \in AG$ *with* $G = \langle N, \Sigma, P, S \rangle$

The Circularity Test II

Algorithm 17.3 (Circularity test for attribute grammars)

Input: $\mathfrak{A} = \langle G, E, V \rangle \in AG$ *with* $G = \langle N, \Sigma, P, S \rangle$

Procedure:

1. *for every* $A \in N$, *iteratively construct* $IS(A)$ *as follows:*
   1. *if* $\pi = A \to w \in P$, *then* $is[\pi] \in IS(A)$
   2. *if* $\pi = A \to w_0 A_1 w_1 \dots A_r w_r \in P$ *and* $is_i \in IS(A_i)$ *for every* $i \in [r]$, *then* $is[\pi; is_1, \dots, is_r] \in IS(A)$

The Circularity Test II

Algorithm 17.3 (Circularity test for attribute grammars)

Input: $\mathfrak{A} = \langle G, E, V \rangle \in AG$ *with* $G = \langle N, \Sigma, P, S \rangle$

Procedure:

1. *for every* $A \in N$, *iteratively construct* $IS(A)$ *as follows:*
   1. *if* $\pi = A \to w \in P$, *then* $is[\pi] \in IS(A)$
   2. *if* $\pi = A \to w_0 A_1 w_1 \ldots A_r w_r \in P$ *and* $is_i \in IS(A_i)$ *for every* $i \in [r]$, *then* $is[\pi; is_1, \ldots, is_r] \in IS(A)$
2. *test whether* $\mathfrak{A}$ *is circular by checking if there exist* $\pi = A \to w_0 A_1 w_1 \ldots A_r w_r \in P$ *and* $is_i \in IS(A_i)$ *for every* $i \in [r]$ *such that the following relation is cyclic:*

   $$\to_\pi \cup \bigcup_{i=1}^{r} \{(\beta.p_i, \alpha.p_i) \mid (\beta, \alpha) \in is_i\}$$

   (where $p_i := \sum_{j=1}^{i} |w_{j-1}| + i$*)*

Algorithm 17.3 (Circularity test for attribute grammars)

Input: $\mathfrak{A} = \langle G, E, V \rangle \in AG$ *with* $G = \langle N, \Sigma, P, S \rangle$

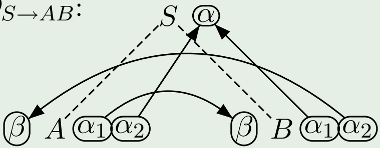
Procedure:

1. *for every* $A \in N$, *iteratively construct* $IS(A)$ *as follows:*
   1. *if* $\pi = A \to w \in P$, *then* $is[\pi] \in IS(A)$
   2. *if* $\pi = A \to w_0 A_1 w_1 \ldots A_r w_r \in P$ *and* $is_i \in IS(A_i)$ *for every* $i \in [r]$, *then* $is[\pi; is_1, \ldots, is_r] \in IS(A)$
2. *test whether* $\mathfrak{A}$ *is circular by checking if there exist* $\pi = A \to w_0 A_1 w_1 \ldots A_r w_r \in P$ *and* $is_i \in IS(A_i)$ *for every* $i \in [r]$ *such that the following relation is cyclic:*
   $$\to_\pi \cup \bigcup_{i=1}^{r} \{(\beta.p_i, \alpha.p_i) \mid (\beta, \alpha) \in is_i\}$$
   (where $p_i := \sum_{j=1}^{i} |w_{j-1}| + i$*)*

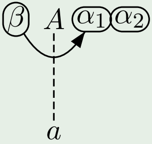
Output: *"yes" or "no"*

Example 17.4

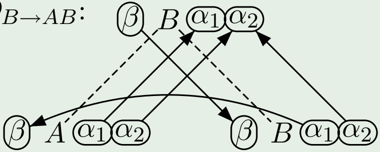
$D_{S \to AB}$:



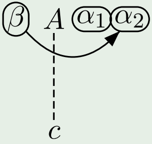
$D_{A \to a}$:



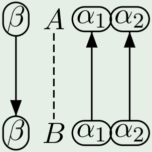
$D_{B \to AB}$:



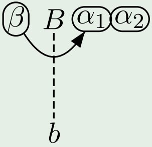
$D_{A \to c}$:



$D_{A \to B}$:



$D_{B \to b}$:



Application of Algorithm 17.3: on the board

Outline

Correctness and Complexity of Circularity Test

Theorem 17.5 (Correctness of circularity test)

An attribute grammar is circular iff Algorithm 17.3 yields the answer "yes".

Correctness and Complexity of Circularity Test

Theorem 17.5 (Correctness of circularity test)

An attribute grammar is circular iff Algorithm 17.3 yields the answer "yes".

Proof.

by induction on the syntax tree t with cyclic D_t □

Correctness and Complexity of Circularity Test

Theorem 17.5 (Correctness of circularity test)

An attribute grammar is circular iff Algorithm 17.3 yields the answer "yes".

Proof.

by induction on the syntax tree t with cyclic D_t □

Lemma 17.6

The time complexity of the circularity test is exponential in the size of the attribute grammar (= maximal length of right-hand sides of productions).

Correctness and Complexity of Circularity Test

Theorem 17.5 (Correctness of circularity test)

An attribute grammar is circular iff Algorithm 17.3 yields the answer "yes".

Proof.

by induction on the syntax tree t with cyclic D_t □

Lemma 17.6

The time complexity of the circularity test is exponential in the size of the attribute grammar (= maximal length of right-hand sides of productions).

Proof.

by reduction of the word problem of alternating Turing machines (see M. Jazayeri: *A Simpler Construction for Showing the Intrinsically Exponential Complexity of the Circularity Problem for Attribute Grammars*, Comm. of the ACM 28(4), 1981, pp. 715–720) □

Outline

Attribute Evaluation Methods

Given:

- noncircular attribute grammar $\mathfrak{A} = \langle G, E, V \rangle \in AG$
- syntax tree t of G
- valuation $v : Syn_\Sigma \to V$ where
  $Syn_\Sigma := \{\alpha.k \mid k \text{ labelled by } a \in \Sigma, \alpha \in \text{syn}(a)\} \subseteq Var_t$

Attribute Evaluation Methods

Given:
- noncircular attribute grammar $\mathfrak{A} = \langle G, E, V \rangle \in AG$
- syntax tree t of G
- valuation $v : Syn_\Sigma \to V$ where
  $Syn_\Sigma := \{\alpha.k \mid k \text{ labelled by } a \in \Sigma, \alpha \in \text{syn}(a)\} \subseteq Var_t$

Goal: extend v to (partial) solution $v : Var_t \to V$

Attribute Evaluation Methods

Given:
- noncircular attribute grammar $\mathfrak{A} = \langle G, E, V \rangle \in AG$
- syntax tree t of G
- valuation $v : Syn_\Sigma \to V$ where
  $Syn_\Sigma := \{\alpha.k \mid k \text{ labelled by } a \in \Sigma, \alpha \in \text{syn}(a)\} \subseteq Var_t$

Goal: extend v to (partial) solution $v : Var_t \to V$

Methods:
1. Topological sorting of D_t:
   1. start with attribute variables which depend at most on synthesized attributes of terminals (Syn_Σ)
   2. proceed by successive substitution

Attribute Evaluation Methods

Given:
- noncircular attribute grammar $\mathfrak{A} = \langle G, E, V \rangle \in AG$
- syntax tree t of G
- valuation $v : Syn_\Sigma \to V$ where $Syn_\Sigma := \{\alpha.k \mid k \text{ labelled by } a \in \Sigma, \alpha \in \text{syn}(a)\} \subseteq Var_t$

Goal: extend v to (partial) solution $v : Var_t \to V$

Methods:
1. Topological sorting of D_t:
   1. start with attribute variables which depend at most on synthesized attributes of terminals (Syn_Σ)
   2. proceed by successive substitution
2. Recursive functions (for strongly noncircular AGs; later):
   1. for every $A \in N$ and $\alpha \in \text{syn}(A)$, define evaluation function $g_{A,\alpha}$ with the following parameters:
      - the node of t where α has to be evaluated and
      - all inherited attributes of A on which α (potentially) depends
   2. for every $\alpha \in \text{syn}(S)$, evaluate $g_{S,\alpha}(k_0)$ where k_0 denotes the root of t

Attribute Evaluation Methods

Given:
- noncircular attribute grammar $\mathfrak{A} = \langle G, E, V \rangle \in AG$
- syntax tree t of G
- valuation $v : Syn_\Sigma \to V$ where
  $Syn_\Sigma := \{\alpha.k \mid k \text{ labelled by } a \in \Sigma, \alpha \in \text{syn}(a)\} \subseteq Var_t$

Goal: extend v to (partial) solution $v : Var_t \to V$

Methods:

1. Topological sorting of D_t:
   1. start with attribute variables which depend at most on synthesized attributes of terminals (Syn_Σ)
   2. proceed by successive substitution
2. Recursive functions (for strongly noncircular AGs; later):
   1. for every $A \in N$ and $\alpha \in \text{syn}(A)$, define evaluation function $g_{A,\alpha}$ with the following parameters:
      - the node of t where α has to be evaluated and
      - all inherited attributes of A on which α (potentially) depends
   2. for every $\alpha \in \text{syn}(S)$, evaluate $g_{S,\alpha}(k_0)$ where k_0 denotes the root of t
3. Special cases: S-attributed grammars (`yacc`), L-attributed grammars

Outline

Attribute Evaluation by Topological Sorting

Algorithm 17.7 (Evaluation by topological sorting)

Input: *noncircular* $\mathfrak{A} = \langle G, E, V \rangle \in AG$, *syntax tree* t *of* G,
valuation $v : Syn_\Sigma \to V$

Attribute Evaluation by Topological Sorting

Algorithm 17.7 (Evaluation by topological sorting)

Input: *noncircular* $\mathfrak{A} = \langle G, E, V \rangle \in AG$, *syntax tree* t *of* G, *valuation* $v : Syn_\Sigma \to V$

Procedure:

1. *let* $Var := Var_t \setminus Syn_\Sigma$ *(* attributes to be evaluated *)*
2. *while* $Var \neq \emptyset$ *do*
   1. *let* $x \in Var$ *such that* $\{y \in Var \mid y \to_t x\} = \emptyset$
   2. *let* $x = f(x_1, \ldots, x_n) \in E_t$
   3. *let* $v(x) := f(v(x_1), \ldots, v(x_n))$
   4. *let* $Var := Var \setminus \{x\}$

Algorithm 17.7 (Evaluation by topological sorting)

Input: *noncircular* $\mathfrak{A} = \langle G, E, V \rangle \in AG$, *syntax tree* t *of* G, *valuation* $v : Syn_\Sigma \to V$

Procedure:

1. *let* $Var := Var_t \setminus Syn_\Sigma$ *(* attributes to be evaluated *)*
2. *while* $Var \neq \emptyset$ *do*
   1. *let* $x \in Var$ *such that* $\{y \in Var \mid y \to_t x\} = \emptyset$
   2. *let* $x = f(x_1, \ldots, x_n) \in E_t$
   3. *let* $v(x) := f(v(x_1), \ldots, v(x_n))$
   4. *let* $Var := Var \setminus \{x\}$

Output: *solution* $v : Var_t \to V$

Algorithm 17.7 (Evaluation by topological sorting)

Input: *noncircular* $\mathfrak{A} = \langle G, E, V \rangle \in AG$, *syntax tree* t *of* G, *valuation* $v : Syn_\Sigma \to V$

Procedure:

1. *let* $Var := Var_t \setminus Syn_\Sigma$ *(* attributes to be evaluated *)*
2. *while* $Var \neq \emptyset$ *do*
    1. *let* $x \in Var$ *such that* $\{y \in Var \mid y \to_t x\} = \emptyset$
    2. *let* $x = f(x_1, \ldots, x_n) \in E_t$
    3. *let* $v(x) := f(v(x_1), \ldots, v(x_n))$
    4. *let* $Var := Var \setminus \{x\}$

Output: *solution* $v : Var_t \to V$

Remark: noncircularity guarantees that in step 2.1 at least one such x is available

Attribute Evaluation by Topological Sorting

Algorithm 17.7 (Evaluation by topological sorting)

Input: *noncircular* $\mathfrak{A} = \langle G, E, V \rangle \in AG$, *syntax tree* t *of* G, *valuation* $v : Syn_\Sigma \to V$

Procedure:
1. *let* $Var := Var_t \setminus Syn_\Sigma$ (* *attributes to be evaluated* *)
2. *while* $Var \neq \emptyset$ *do*
   1. *let* $x \in Var$ *such that* $\{y \in Var \mid y \to_t x\} = \emptyset$
   2. *let* $x = f(x_1, \ldots, x_n) \in E_t$
   3. *let* $v(x) := f(v(x_1), \ldots, v(x_n))$
   4. *let* $Var := Var \setminus \{x\}$

Output: *solution* $v : Var_t \to V$

Remark: noncircularity guarantees that in step 2.1 at least one such x is available

Example 17.8

see Examples 15.1 and 15.2 (Knuth's binary numbers)