

Compiler Construction

Lecture 16: Semantic Analysis II (Circularity of Attribute Grammars)

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

- 1 Repetition: Attribute Grammars
- 2 The Attribute Equation System
- 3 Circularity of Attribute Grammars
- 4 Attribute Dependency Graphs
- 5 Testing Attribute Grammars for Circularity
- 6 The Circularity Test
- 7 Correctness and Complexity of the Circularity Test

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**

$\implies$ **Semantic analysis = attribute evaluation**

Result: **attributed syntax tree**

In greater detail:

- With every nonterminal 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.

Binary Numbers with Inherited Attributes I

Example (synthesized + inherited attributes)

Binary numbers (with fraction):

G'_B : Numbers $S \rightarrow L$

$S \rightarrow L.L$

Lists $L \rightarrow B$

$L \rightarrow LB$

Bits $B \rightarrow 0$

Bits $B \rightarrow 1$

Binary Numbers with Inherited Attributes I

Example (synthesized + inherited attributes)

Binary numbers (with fraction):

G'_B : Numbers	$S \rightarrow L$	$v.0 = v.1$ $p.1 = 0$
	$S \rightarrow L.L$	$v.0 = v.1 + v.3$ $p.1 = 0$ $p.3 = -l.3$
	Lists $L \rightarrow B$	$v.0 = v.1$ $l.0 = 1$ $p.1 = p.0$
		$L \rightarrow LB$ $v.0 = v.1 + v.2$ $l.0 = l.1 + 1$ $p.1 = p.0 + 1$ $p.2 = p.0$
Bits	$B \rightarrow 0$	$v.0 = 0$
Bits	$B \rightarrow 1$	$v.0 = 2^{p.0}$

Synthesized attributes of S, L, B : v (value; domain: $V^v := \mathbb{Q}$)

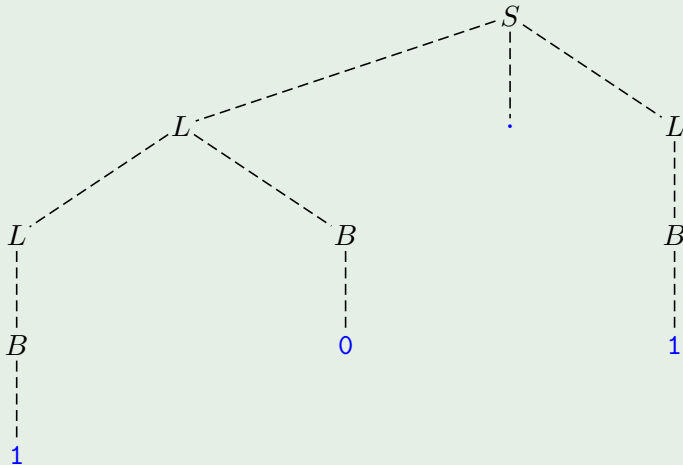
of L : l (length; domain: $V^l := \mathbb{N}$)

Inherited attribute of L, B : p (position; domain: $V^p := \mathbb{Z}$)

Binary Numbers with Inherited Attributes II

Example (continued)

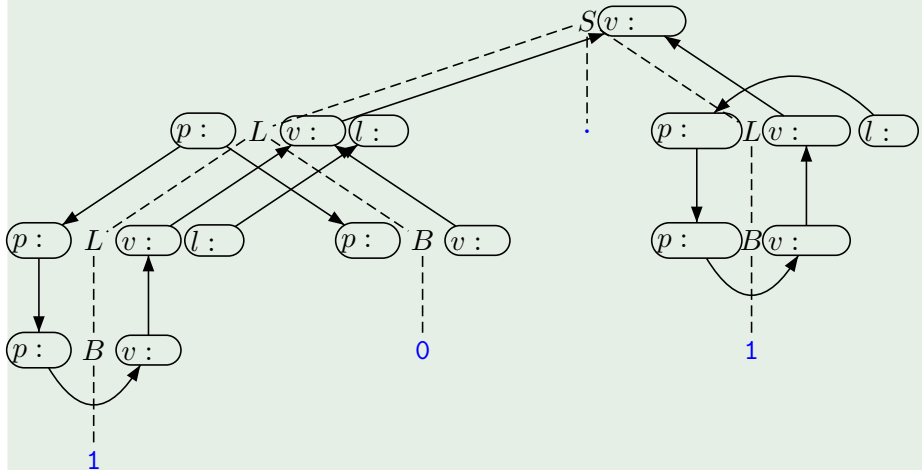
Syntax tree for 10.1:



Binary Numbers with Inherited Attributes II

Example (continued)

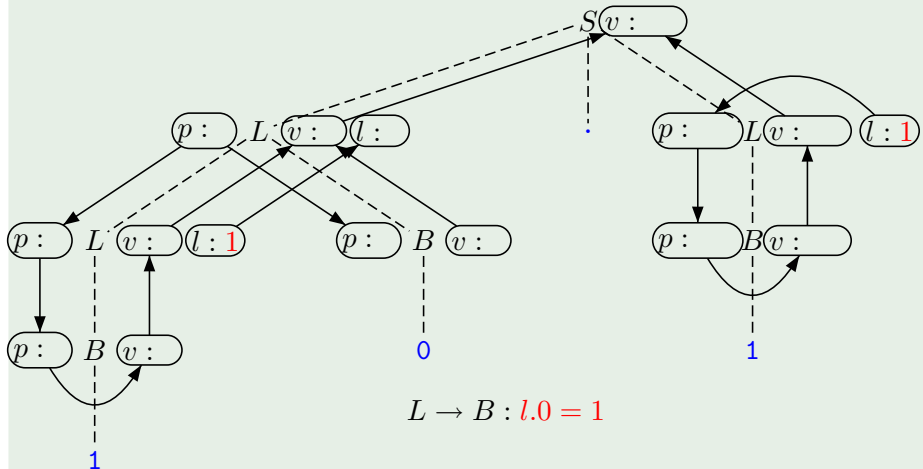
Attributed syntax tree for 10.1:



Binary Numbers with Inherited Attributes II

Example (continued)

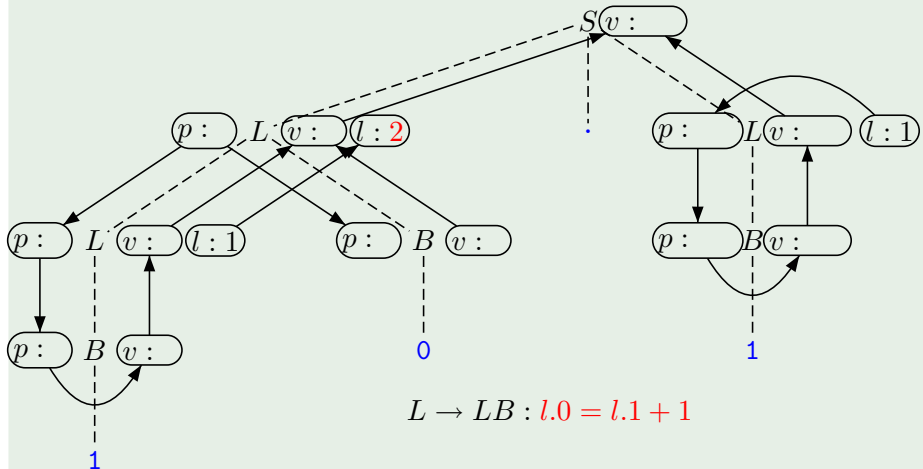
Attributed syntax tree for 10.1:



Binary Numbers with Inherited Attributes II

Example (continued)

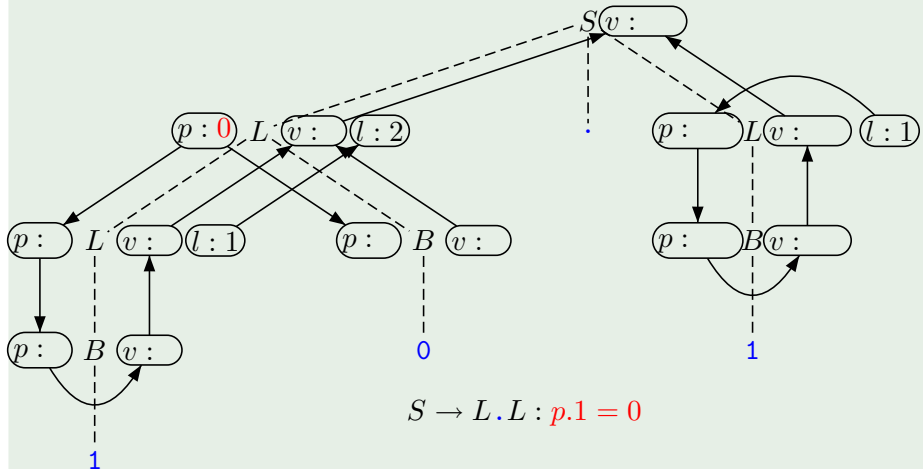
Attributed syntax tree for 10.1:



Binary Numbers with Inherited Attributes II

Example (continued)

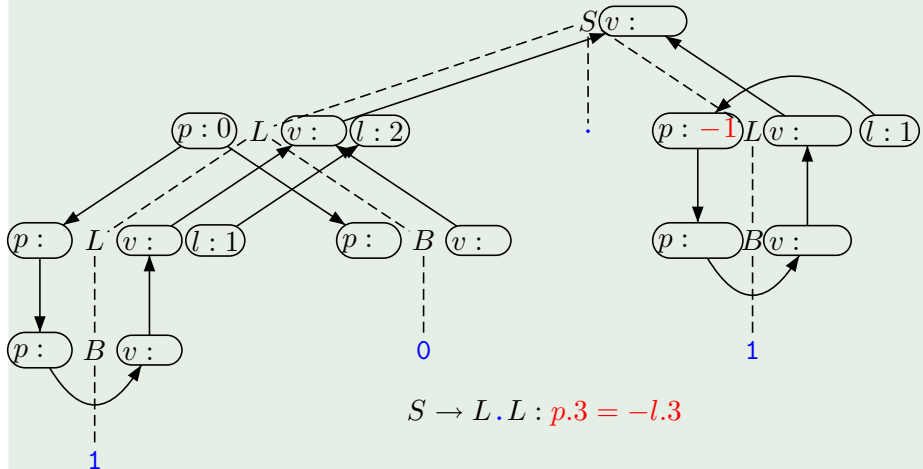
Attributed syntax tree for 10.1:



Binary Numbers with Inherited Attributes II

Example (continued)

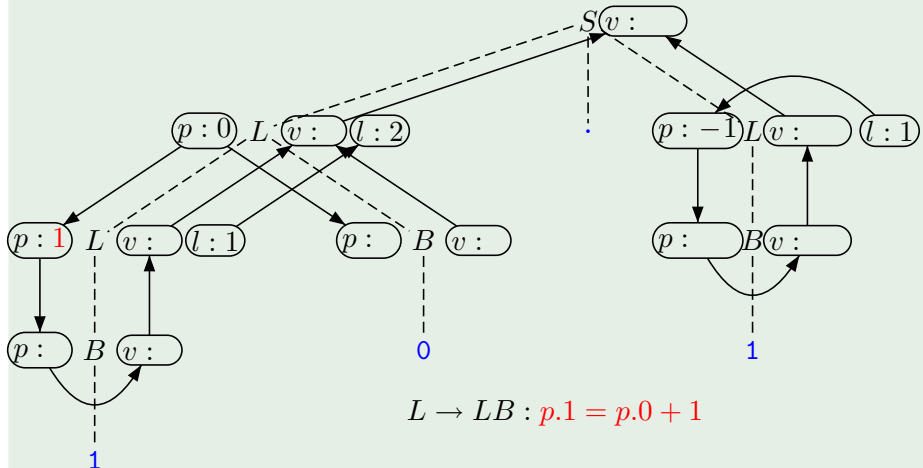
Attributed syntax tree for 10.1:



Binary Numbers with Inherited Attributes II

Example (continued)

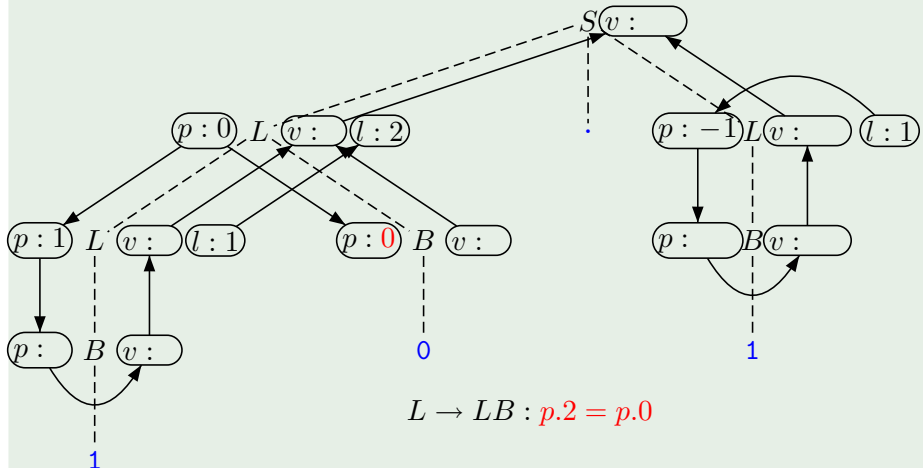
Attributed syntax tree for 10.1:



Binary Numbers with Inherited Attributes II

Example (continued)

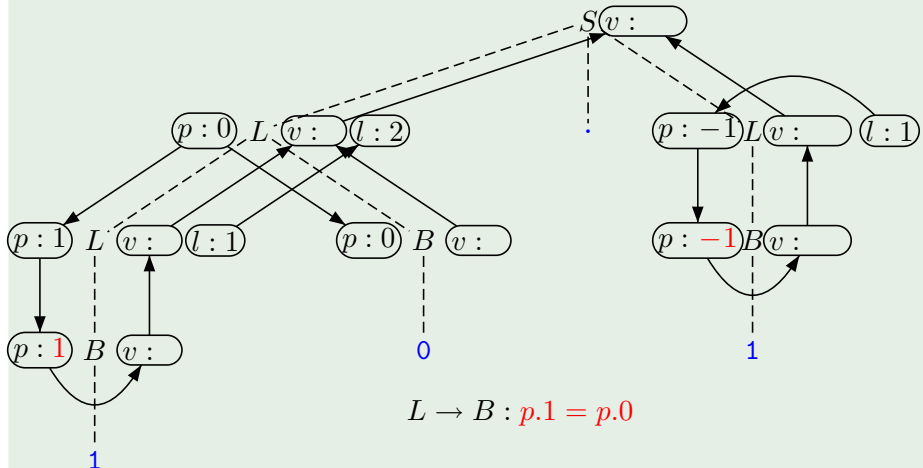
Attributed syntax tree for 10.1:



Binary Numbers with Inherited Attributes II

Example (continued)

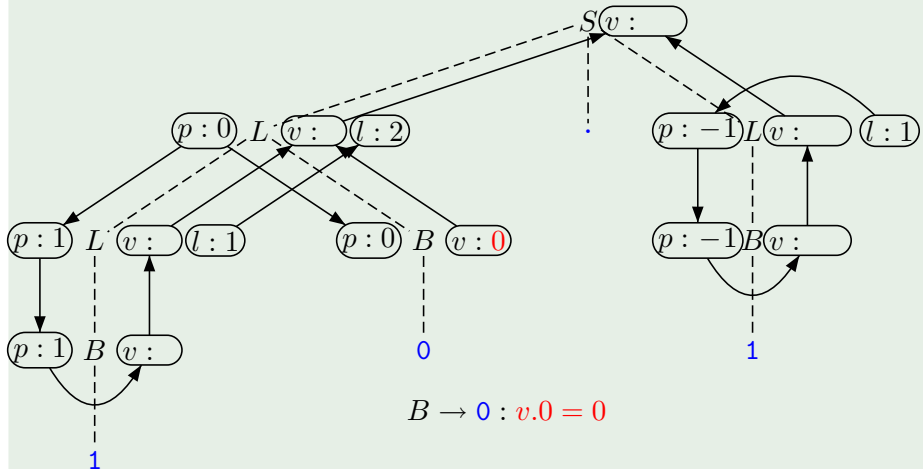
Attributed syntax tree for 10.1:



Binary Numbers with Inherited Attributes II

Example (continued)

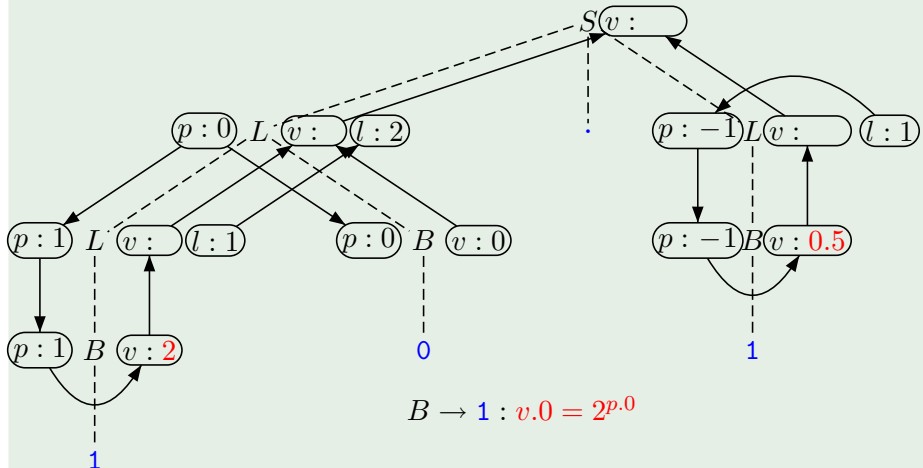
Attributed syntax tree for 10.1:



Binary Numbers with Inherited Attributes II

Example (continued)

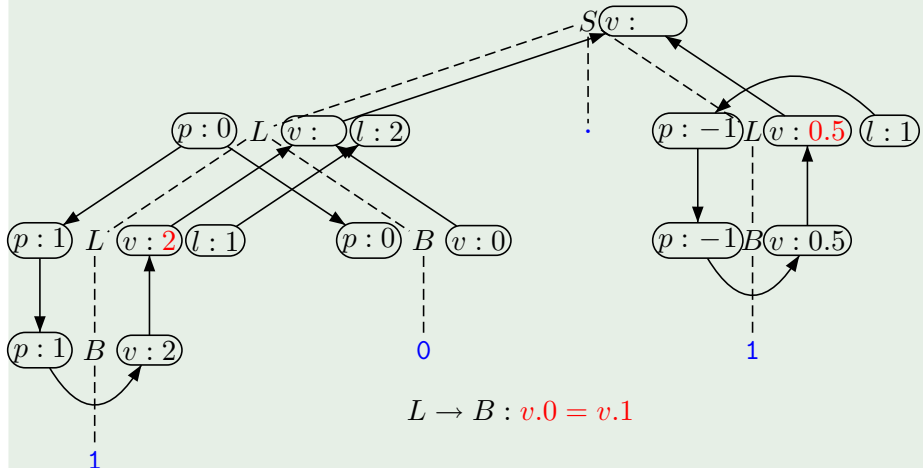
Attributed syntax tree for 10.1:



Binary Numbers with Inherited Attributes II

Example (continued)

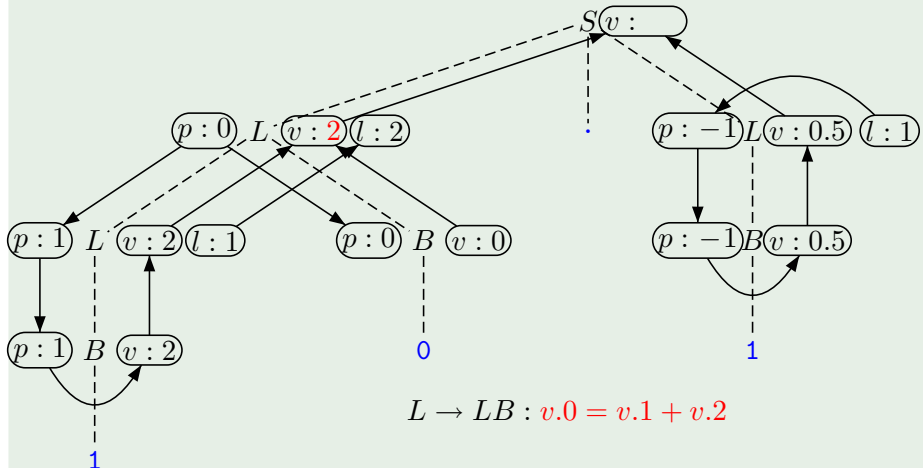
Attributed syntax tree for 10.1:



Binary Numbers with Inherited Attributes II

Example (continued)

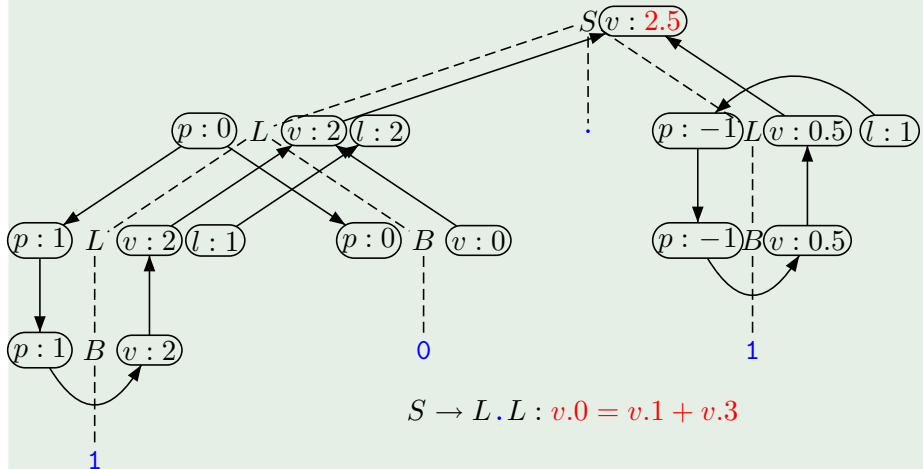
Attributed syntax tree for 10.1:



Binary Numbers with Inherited Attributes II

Example (continued)

Attributed syntax tree for 10.1:



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 $att : X \rightarrow 2^{Att}$ be an attribute assignment, and let $syn(Y) := att(Y) \cap Syn$ and $inh(Y) := att(Y) \cap Inh$ 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:
 $In_\pi := \{\alpha.i \mid (i = 0, \alpha \in syn(Y_i)) \text{ or } (i \in [r], \alpha \in inh(Y_i))\}$
 $Out_\pi := Var_\pi \setminus In_\pi$
- 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 $f : V^{\alpha_1} \times \dots \times V^{\alpha_n} \rightarrow V^\alpha$.
- 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: $\mathfrak{A} \in AG$.

- 1 Repetition: Attribute Grammars
- 2 The Attribute Equation System
- 3 Circularity of Attribute Grammars
- 4 Attribute Dependency Graphs
- 5 Testing Attribute Grammars for Circularity
- 6 The Circularity Test
- 7 Correctness and Complexity of the Circularity Test

Definition 16.1 (Attribution of syntax trees)

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

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

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

Definition 16.1 (Attribution of syntax trees)

Let $\mathfrak{A} = \langle G, E, V \rangle \in AG$, 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 .

Definition 16.1 (Attribution of syntax trees)

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

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

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

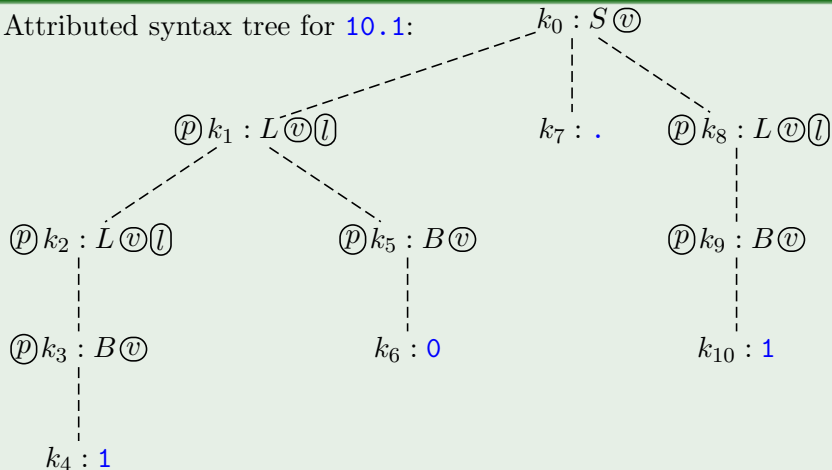
- Let $k_0 \in K$ be an (inner) node where production $\pi = Y_0 \rightarrow Y_1 \dots Y_r \in P$ is applied, and let $k_1, \dots, k_r \in K$ be the corresponding successor nodes. The **attribute equation system** E_{k_0} of k_0 is obtained from E_π by substituting every attribute index $i \in \{0, \dots,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

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

Attribution of Syntax Trees II

Example 16.2 (cf. Example 16.1)

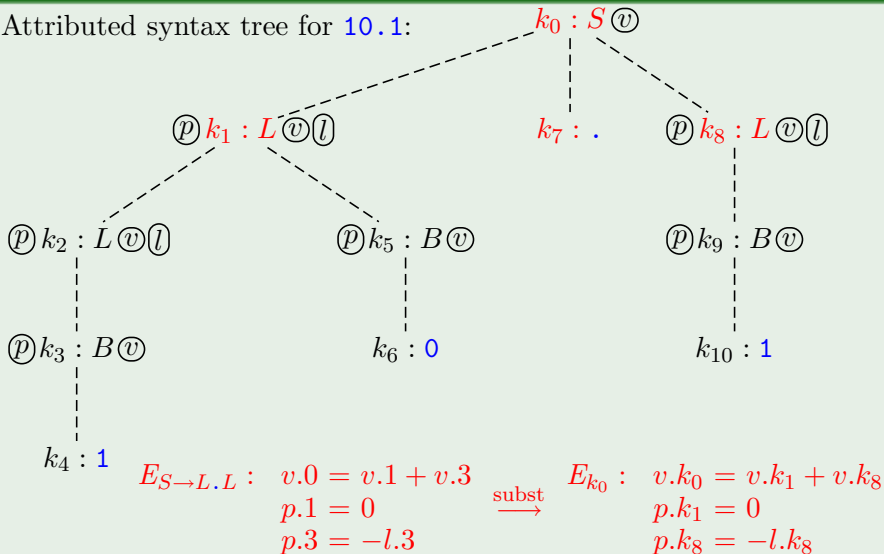
Attributed syntax tree for 10.1:



Attribution of Syntax Trees II

Example 16.2 (cf. Example 16.1)

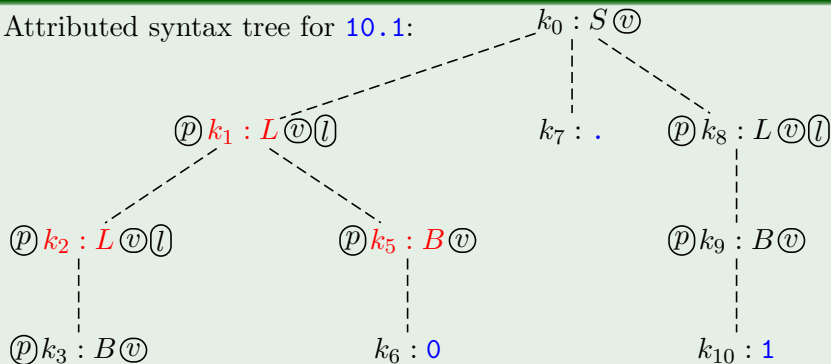
Attributed syntax tree for 10.1:



Attribution of Syntax Trees II

Example 16.2 (cf. Example 16.1)

Attributed syntax tree for 10.1:



$k_4 : 1$

$$E_{L \rightarrow LB} : \quad \begin{aligned} v.0 &= v.1 + v.2 \\ l.0 &= l.1 + 1 \\ p.1 &= p.0 + 1 \\ p.2 &= p.0 \end{aligned}$$

subst
→

$$E_{k_1} : \quad \begin{aligned} v.k_1 &= v.k_2 + v.k_5 \\ l.k_1 &= l.k_2 + 1 \\ p.k_2 &= p.k_1 + 1 \\ p.k_5 &= p.k_1 \end{aligned}$$

Corollary 16.3

For each $\alpha.k \in \text{Var}_t$ except 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 $\alpha.k$.

Corollary 16.3

For each $\alpha.k \in \text{Var}_t$ except 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.

- 1 Repetition: Attribute Grammars
- 2 The Attribute Equation System
- 3 Circularity of Attribute Grammars**
- 4 Attribute Dependency Graphs
- 5 Testing Attribute Grammars for Circularity
- 6 The Circularity Test
- 7 Correctness and Complexity of the Circularity Test

Definition 16.4 (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 $\alpha.k \in Var_t$ and $\alpha.k = f(\alpha.k_1, \dots, \alpha.k_n) \in E_t$,

$$v(\alpha.k) = f(v(\alpha.k_1), \dots, v(\alpha.k_n)).$$

Definition 16.4 (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 $\alpha.k \in Var_t$ and $\alpha.k = f(\alpha.k_1, \dots, \alpha.k_n) \in E_t$,

$$v(\alpha.k) = f(v(\alpha.k_1), \dots, v(\alpha.k_n)).$$

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

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

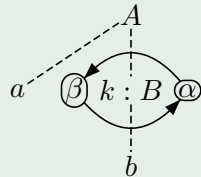
Example 16.5

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

Example 16.5

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

$\Rightarrow$ cyclic dependency:



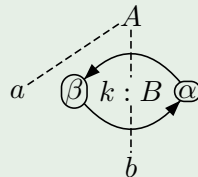
$$E_t : \begin{aligned} \beta.k &= f(\alpha.k) \\ \alpha.k &= g(\beta.k) \end{aligned}$$

Solvability of Attribute Equation System II

Example 16.5

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

$\Rightarrow$ cyclic dependency:



$\Rightarrow$ for $V^\alpha := V^\beta := \mathbb{N}$, $g(x) := x$, 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 $y \in \mathbb{N}$)

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

Goal: unique solvability of equation system
 $\implies$ avoid cyclic dependencies

Circularity of Attribute Grammars

Goal: **unique solvability** of equation system
 $\implies$ avoid cyclic dependencies

Definition 16.6 (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**.

Circularity of Attribute Grammars

Goal: **unique solvability** of equation system
 $\implies$ avoid cyclic dependencies

Definition 16.6 (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 later).

- 1 Repetition: Attribute Grammars
- 2 The Attribute Equation System
- 3 Circularity of Attribute Grammars
- 4 Attribute Dependency Graphs**
- 5 Testing Attribute Grammars for Circularity
- 6 The Circularity Test
- 7 Correctness and Complexity of the Circularity Test

Goal: graphic representation of attribute dependencies

Definition 16.7 (Production dependency graph)

Let $\mathfrak{A} = \langle G, E, V \rangle \in AG$ with $G = \langle N, \Sigma, P, S \rangle$. Every production $\pi \in P$ determines the **dependency graph** $D_\pi := \langle Var_\pi, \rightarrow_\pi \rangle$ where the set of edges $\rightarrow_\pi \subseteq Var_\pi \times Var_\pi$ is given by

$$x \rightarrow_\pi y \quad \text{iff} \quad y = f(\dots, x, \dots) \in E_\pi.$$

Attribute Dependency Graphs I

Goal: graphic representation of attribute dependencies

Definition 16.7 (Production dependency graph)

Let $\mathfrak{A} = \langle G, E, V \rangle \in AG$ with $G = \langle N, \Sigma, P, S \rangle$. Every production $\pi \in P$ determines the **dependency graph** $D_\pi := \langle Var_\pi, \rightarrow_\pi \rangle$ where the set of edges $\rightarrow_\pi \subseteq Var_\pi \times Var_\pi$ is given by

$$x \rightarrow_\pi y \quad \text{iff} \quad y = f(\dots, x, \dots) \in E_\pi.$$

Corollary 16.8

*The dependency graph of a production is acyclic
(since $\rightarrow_\pi \subseteq Out_\pi \times In_\pi$).*

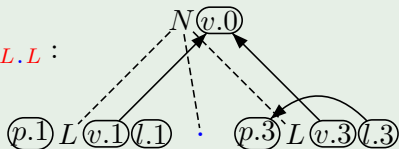
Example 16.9 (cf. Example 16.1)

① $N \rightarrow L.L :$ $\Rightarrow D_{N \rightarrow L.L} :$

$$v.0 = v.1 + v.3$$

$$p.1 = 0$$

$$p.3 = -l.3$$



Attribute Dependency Graphs II

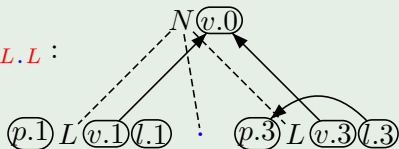
Example 16.9 (cf. Example 16.1)

① $N \rightarrow L.L : \quad \Rightarrow \quad D_{N \rightarrow L.L} :$

$$v.0 = v.1 + v.3$$

$$p.1 = 0$$

$$p.3 = -l.3$$



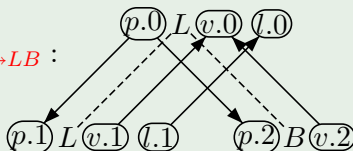
② $L \rightarrow LB : \quad \Rightarrow \quad D_{L \rightarrow LB} :$

$$v.0 = v.1 + v.2$$

$$l.0 = l.1 + 1$$

$$p.1 = p.0 + 1$$

$$p.2 = p.0$$



Attribute Dependency Graphs III

Just as the attribute equation system E_t of a syntax tree t is obtained from the semantic rules of the contributing productions, the dependency graph of t is obtained by “glueing together” the dependency graphs of the productions.

Attribute Dependency Graphs III

Just as the attribute equation system E_t of a syntax tree t is obtained from the semantic rules of the contributing productions, the dependency graph of t is obtained by “glueing together” the dependency graphs of the productions.

Definition 16.10 (Tree dependency graph)

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

- The **dependency graph** of t is defined by $D_t := \langle Var_t, \rightarrow_t \rangle$ where the set of edges $\rightarrow_t \subseteq Var_t \times Var_t$ is given by
$$x \rightarrow_t y \quad \text{iff} \quad y = f(\dots, x, \dots) \in E_t.$$
- D_t is called **cyclic** if there exists $x \in Var_t$ such that $x \rightarrow_t^+ x$.

Attribute Dependency Graphs III

Just as the attribute equation system E_t of a syntax tree t is obtained from the semantic rules of the contributing productions, the dependency graph of t is obtained by “glueing together” the dependency graphs of the productions.

Definition 16.10 (Tree dependency graph)

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

- The **dependency graph** of t is defined by $D_t := \langle Var_t, \rightarrow_t \rangle$ where the set of edges $\rightarrow_t \subseteq Var_t \times Var_t$ is given by
$$x \rightarrow_t y \quad \text{iff} \quad y = f(\dots, x, \dots) \in E_t.$$
- D_t is called **cyclic** if there exists $x \in Var_t$ such that $x \rightarrow_t^+ x$.

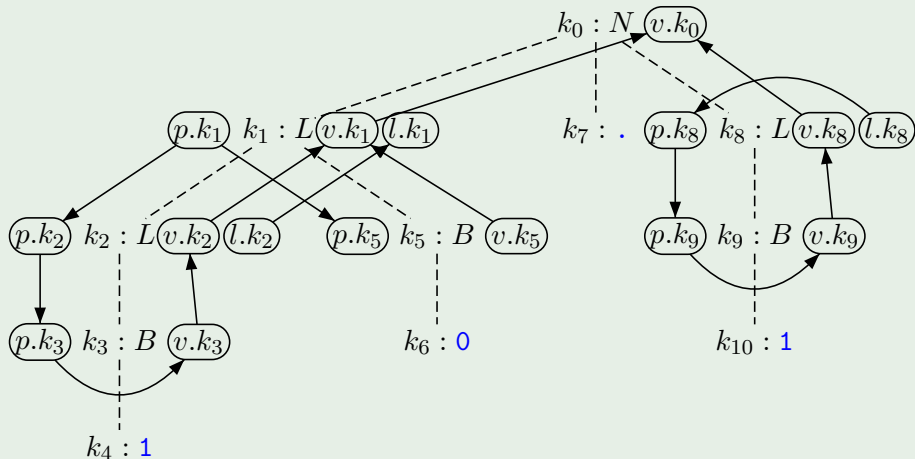
Corollary 16.11

An attribute grammar $\mathfrak{A} = \langle G, E, V \rangle \in AG$ is **circular** iff there exists a syntax tree t of G such that D_t is **cyclic**.

Attribute Dependency Graphs IV

Example 16.12 (cf. Example 16.1)

(Acyclic) dependency graph of the syntax tree for 10.1:



- 1 Repetition: Attribute Grammars
- 2 The Attribute Equation System
- 3 Circularity of Attribute Grammars
- 4 Attribute Dependency Graphs
- 5 Testing Attribute Grammars for Circularity
- 6 The Circularity Test
- 7 Correctness and Complexity of the Circularity Test

Observation: a cycle in the dependency graph D_t of a given syntax tree t is caused by the occurrence of a “cover” production

$\pi = A_0 \rightarrow w_0 A_1 w_1 \dots A_r w_r \in P$ 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 $\text{inh}(A_i)$.

Observation: a cycle in the dependency graph D_t of a given syntax tree t is caused by the occurrence of a “cover” production

$\pi = A_0 \rightarrow w_0 A_1 w_1 \dots A_r w_r \in P$ 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 16.13

on the board

Observation: a cycle in the dependency graph D_t of a given syntax tree t is caused by the occurrence of a “cover” production

$\pi = A_0 \rightarrow w_0 A_1 w_1 \dots A_r w_r \in P$ 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 16.13

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 $\text{inh}(A_i)$.

Definition 16.14 (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 $\beta.k \rightarrow_t^+ \alpha.k$, then α is dependent on β below A in t (notation: $\beta \xrightarrow{A}_t \alpha$).

Definition 16.14 (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 $\beta.k \rightarrow_t^+ \alpha.k$, then **α is dependent on β below A in t** (notation: $\beta \xrightarrow{A} \alpha$).
- For every syntax tree t with root label $A \in N$,
$$\text{is}(A, t) := \{(\beta, \alpha) \in \text{inh}(A) \times \text{syn}(A) \mid \beta \xrightarrow{A} \alpha \text{ in } t\}.$$

Definition 16.14 (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 $\beta.k \rightarrow_t^+ \alpha.k$, then α is dependent on β below A in t (notation: $\beta \xrightarrow{A} \alpha$).
- For every syntax tree t with root label $A \in N$,
$$\text{is}(A, t) := \{(\beta, \alpha) \in \text{inh}(A) \times \text{syn}(A) \mid \beta \xrightarrow{A} \alpha \text{ in } t\}.$$
- For every $A \in N$,
$$\text{IS}(A) := \{\text{is}(A, t) \mid t \text{ syntax tree with root label } A\} \\ \subseteq 2^{\text{Inh} \times \text{Syn}}.$$

Definition 16.14 (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 $\beta.k \rightarrow_t^+ \alpha.k$, then α is dependent on β below A in t (notation: $\beta \xrightarrow{A} \alpha$).
- For every syntax tree t with root label $A \in N$,

$$\text{is}(A, t) := \{(\beta, \alpha) \in \text{inh}(A) \times \text{syn}(A) \mid \beta \xrightarrow{A} \alpha \text{ in } t\}.$$
- For every $A \in N$,

$$\text{IS}(A) := \{\text{is}(A, t) \mid t \text{ syntax tree with root label } A\} \\ \subseteq 2^{\text{Inh} \times \text{Syn}}.$$

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

Definition 16.14 (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 $\beta.k \rightarrow_t^+ \alpha.k$, then α is dependent on β below A in t (notation: $\beta \xrightarrow{A} \alpha$).
- For every syntax tree t with root label $A \in N$,
$$\text{is}(A, t) := \{(\beta, \alpha) \in \text{inh}(A) \times \text{syn}(A) \mid \beta \xrightarrow{A} \alpha \text{ in } t\}.$$
- For every $A \in N$,
$$\text{IS}(A) := \{\text{is}(A, t) \mid t \text{ syntax tree with root label } A\} \\ \subseteq 2^{\text{Inh} \times \text{Syn}}.$$

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

Example 16.15

on the board

- 1 Repetition: Attribute Grammars
- 2 The Attribute Equation System
- 3 Circularity of Attribute Grammars
- 4 Attribute Dependency Graphs
- 5 Testing Attribute Grammars for Circularity
- 6 The Circularity Test**
- 7 Correctness and Complexity of the Circularity Test

The Circularity Test I

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

Definition 16.16

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

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

be given by

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

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

The Circularity Test I

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

Definition 16.16

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

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

be given by

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

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

Example 16.17

on the board

The Circularity Test II

Algorithm 16.18 (Circularity test for attribute grammars)

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

Algorithm 16.18 (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 $\pi = A \rightarrow w \in P$, then $is[\pi] \in IS(A)$
- ② if $\pi = A \rightarrow w_0 A_1 w_1 \dots A_r w_r \in P$ 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 16.18 (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 $\pi = A \rightarrow w \in P$, then $is[\pi] \in IS(A)$
- ② if $\pi = A \rightarrow w_0 A_1 w_1 \dots A_r w_r \in P$ and $is_i \in IS(A_i)$ for every $i \in [r]$, then $is[\pi; is_1, \dots, is_r] \in IS(A)$

- ② *test whether $\mathfrak{A}$ is circular* by checking if there exist $\pi = A \rightarrow w_0 A_1 w_1 \dots A_r w_r \in P$ and $is_i \in IS(A_i)$ for every $i \in [r]$ such that the following relation is cyclic:

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

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

The Circularity Test II

Algorithm 16.18 (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 $\pi = A \rightarrow w \in P$, then $is[\pi] \in IS(A)$
- ② if $\pi = A \rightarrow w_0 A_1 w_1 \dots A_r w_r \in P$ and $is_i \in IS(A_i)$ for every $i \in [r]$, then $is[\pi; is_1, \dots, is_r] \in IS(A)$

- ② *test whether $\mathfrak{A}$ is circular* by checking if there exist $\pi = A \rightarrow w_0 A_1 w_1 \dots A_r w_r \in P$ and $is_i \in IS(A_i)$ for every $i \in [r]$ such that the following relation is cyclic:

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

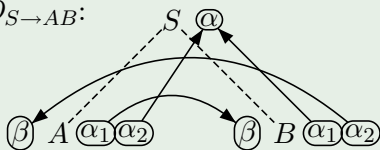
(where $p_i := \sum_{j=1}^i |w_{j-1}| + i$)

Output: “yes” or “no”

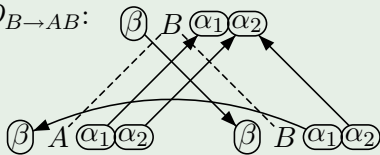
The Circularity Test III

Example 16.19

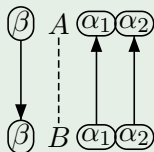
$D_{S \rightarrow AB}$:



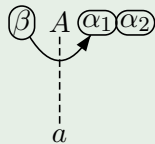
$D_{B \rightarrow AB}$:



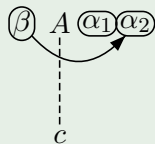
$D_{A \rightarrow B}$:



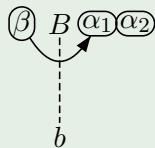
$D_{A \rightarrow a}$:



$D_{A \rightarrow c}$:



$D_{B \rightarrow b}$:



Application of Algorithm 16.18: on the board

- 1 Repetition: Attribute Grammars
- 2 The Attribute Equation System
- 3 Circularity of Attribute Grammars
- 4 Attribute Dependency Graphs
- 5 Testing Attribute Grammars for Circularity
- 6 The Circularity Test
- 7 Correctness and Complexity of the Circularity Test

Theorem 16.20 (Correctness of circularity test)

An attribute grammar is circular iff Algorithm 16.18 yields the answer “yes”.

Correctness and Complexity of Circularity Test

Theorem 16.20 (Correctness of circularity test)

An attribute grammar is circular iff Algorithm 16.18 yields the answer “yes”.

Proof.

by induction on the syntax tree t with cyclic D_t



Correctness and Complexity of Circularity Test

Theorem 16.20 (Correctness of circularity test)

An attribute grammar is circular iff Algorithm 16.18 yields the answer “yes”.

Proof.

by induction on the syntax tree t with cyclic D_t



Lemma 16.21

*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 16.20 (Correctness of circularity test)

An attribute grammar is circular iff Algorithm 16.18 yields the answer “yes”.

Proof.

by induction on the syntax tree t with cyclic D_t □

Lemma 16.21

*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) □