

Compiler Construction

Lecture 12: Semantic Analysis I (Attribute Grammars)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)



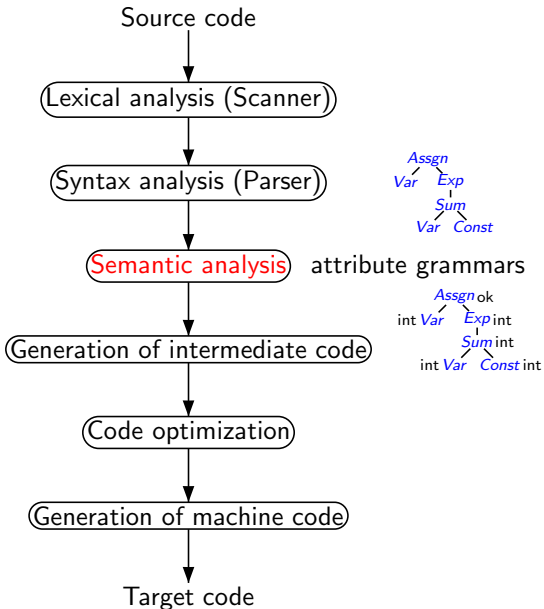
noll@cs.rwth-aachen.de

<http://moves.rwth-aachen.de/teaching/ss-14/cc14/>

Summer Semester 2014

- 1 Overview
- 2 Semantic Analysis
- 3 Attribute Grammars
- 4 Adding Inherited Attributes
- 5 Formal Definition of Attribute Grammars
- 6 The Attribute Equation System

Conceptual Structure of a Compiler



- 1 Overview
- 2 Semantic Analysis**
- 3 Attribute Grammars
- 4 Adding Inherited Attributes
- 5 Formal Definition of Attribute Grammars
- 6 The Attribute Equation System

To generate (efficient) code, the compiler needs to answer many **questions**:

- Are there identifiers that are not declared? Declared but not used?

To generate (efficient) code, the compiler needs to answer many **questions**:

- Are there identifiers that are not declared? Declared but not used?
- Is `x` a scalar, an array, or a procedure? Of which type?

To generate (efficient) code, the compiler needs to answer many **questions**:

- Are there identifiers that are not declared? Declared but not used?
- Is `x` a scalar, an array, or a procedure? Of which type?
- Which declaration of `x` is used by each reference?

To generate (efficient) code, the compiler needs to answer many **questions**:

- Are there identifiers that are not declared? Declared but not used?
- Is `x` a scalar, an array, or a procedure? Of which type?
- Which declaration of `x` is used by each reference?
- Is `x` defined before it is used?

To generate (efficient) code, the compiler needs to answer many **questions**:

- Are there identifiers that are not declared? Declared but not used?
- Is x a scalar, an array, or a procedure? Of which type?
- Which declaration of x is used by each reference?
- Is x defined before it is used?
- Is the expression $3 * x + y$ type consistent?

To generate (efficient) code, the compiler needs to answer many **questions**:

- Are there identifiers that are not declared? Declared but not used?
- Is x a scalar, an array, or a procedure? Of which type?
- Which declaration of x is used by each reference?
- Is x defined before it is used?
- Is the expression $3 * x + y$ type consistent?
- Where should the value of x be stored (register/stack/heap)?

To generate (efficient) code, the compiler needs to answer many **questions**:

- Are there identifiers that are not declared? Declared but not used?
- Is x a scalar, an array, or a procedure? Of which type?
- Which declaration of x is used by each reference?
- Is x defined before it is used?
- Is the expression $3 * x + y$ type consistent?
- Where should the value of x be stored (register/stack/heap)?
- Do p and q refer to the same memory location (aliasing)?
- ...

To generate (efficient) code, the compiler needs to answer many **questions**:

- Are there identifiers that are not declared? Declared but not used?
- Is x a scalar, an array, or a procedure? Of which type?
- Which declaration of x is used by each reference?
- Is x defined before it is used?
- Is the expression $3 * x + y$ type consistent?
- Where should the value of x be stored (register/stack/heap)?
- Do p and q refer to the same memory location (aliasing)?
- ...

These cannot be expressed using context-free grammars!

To generate (efficient) code, the compiler needs to answer many **questions**:

- **Are there identifiers that are not declared?** Declared but not used?
- Is x a scalar, an array, or a procedure? Of which type?
- Which declaration of x is used by each reference?
- Is x defined before it is used?
- Is the expression $3 * x + y$ type consistent?
- Where should the value of x be stored (register/stack/heap)?
- Do p and q refer to the same memory location (aliasing)?
- ...

These cannot be expressed using context-free grammars!

(e.g., $\{ww \mid w \in \Sigma^*\} \notin CFL_{\Sigma}$)

Static semantics

Static semantics refers to properties of program constructs

- which are true for every occurrence of this construct in every program execution (**static**) and
- can be decided at compile time
- but are context-sensitive and thus not expressible using context-free grammars (**semantics**).

Static semantics

Static semantics refers to properties of program constructs

- which are true for every occurrence of this construct in every program execution (**static**) and
- can be decided at compile time
- but are context-sensitive and thus not expressible using context-free grammars (**semantics**).

Example properties

Static: type or declaredness of an identifier, number of registers required to evaluate an expression, ...

Dynamic: value of an expression, size of runtime stack, ...

- 1 Overview
- 2 Semantic Analysis
- 3 Attribute Grammars**
- 4 Adding Inherited Attributes
- 5 Formal Definition of Attribute Grammars
- 6 The Attribute Equation System

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**

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 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.

Advantage: attribute grammars provide a very flexible and broadly applicable mechanism for transporting information through the syntax tree (“syntax-directed translation”)

- Attribute values: symbol tables, data types, code, error flags, ...
- Application in Compiler Construction:
 - static semantics
 - program analysis for optimization
 - code generation
 - error handling
- Automatic attribute evaluation by compiler generators (cf. yacc’s synthesized attributes)
- Originally designed by D. Knuth for defining the **semantics of context-free languages** (Math. Syst. Theory 2 (1968), pp. 127–145)

Example 12.1 (only synthesized 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	$B \rightarrow 1$

Example: Knuth's Binary Numbers I

Example 12.1 (only synthesized attributes)

Binary numbers (with fraction):

G_B :	Numbers	$S \rightarrow L$	$d.0 = d.1$
		$S \rightarrow L.L$	$d.0 = d.1 + d.3/2^{l.3}$
	Lists	$L \rightarrow B$	$d.0 = d.1$
			$l.0 = 1$
		$L \rightarrow LB$	$d.0 = 2 * d.1 + d.2$
			$l.0 = l.1 + 1$
	Bits	$B \rightarrow 0$	$d.0 = 0$
	Bits	$B \rightarrow 1$	$d.0 = 1$

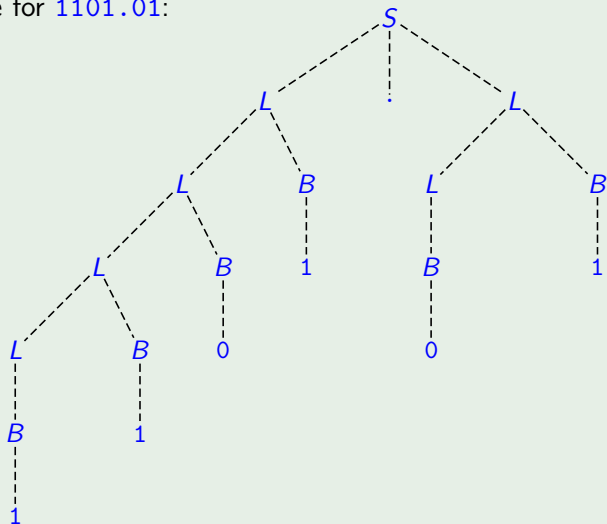
Synthesized attributes of S, L, B : d (decimal value; domain: $V^d := \mathbb{Q}$)
of L : l (length; domain: $V^l := \mathbb{N}$)

Semantic rules: equations with attribute variables
(index = position of symbol; 0 = left-hand side)

Example: Knuth's Binary Numbers II

Example 12.1 (continued)

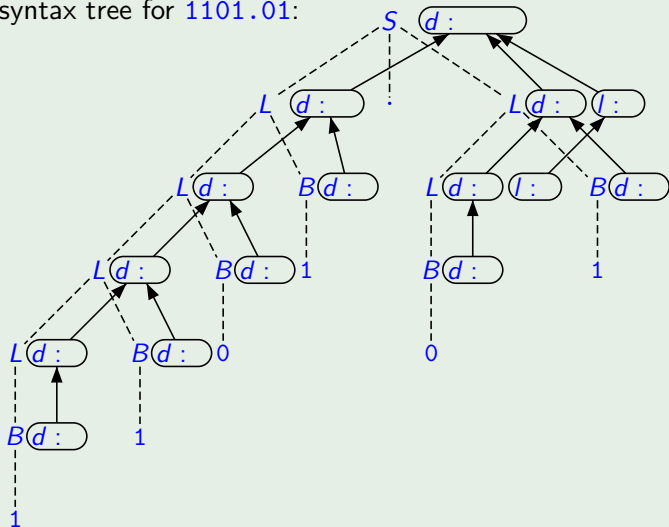
Syntax tree for 1101.01:



Example: Knuth's Binary Numbers II

Example 12.1 (continued)

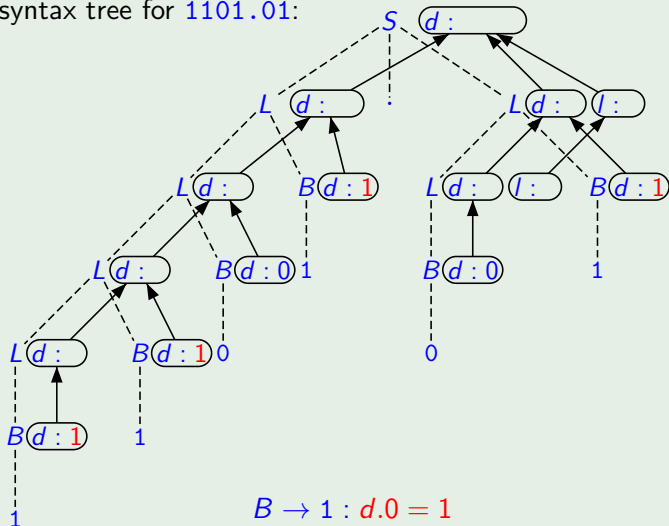
Attributed syntax tree for 1101.01:



Example: Knuth's Binary Numbers II

Example 12.1 (continued)

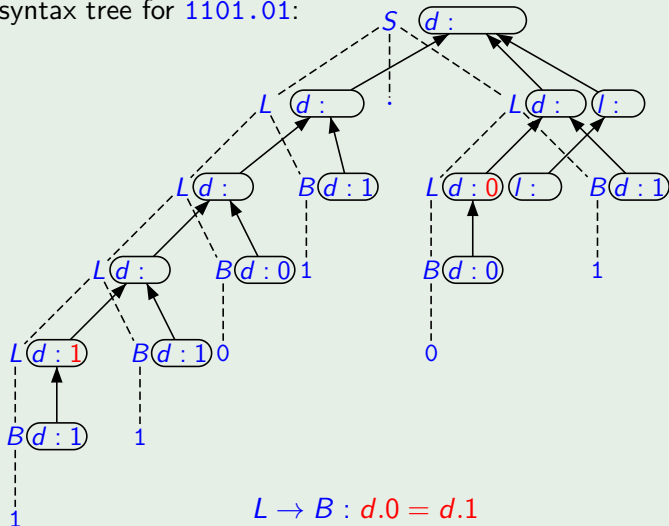
Attributed syntax tree for 1101.01:



Example: Knuth's Binary Numbers II

Example 12.1 (continued)

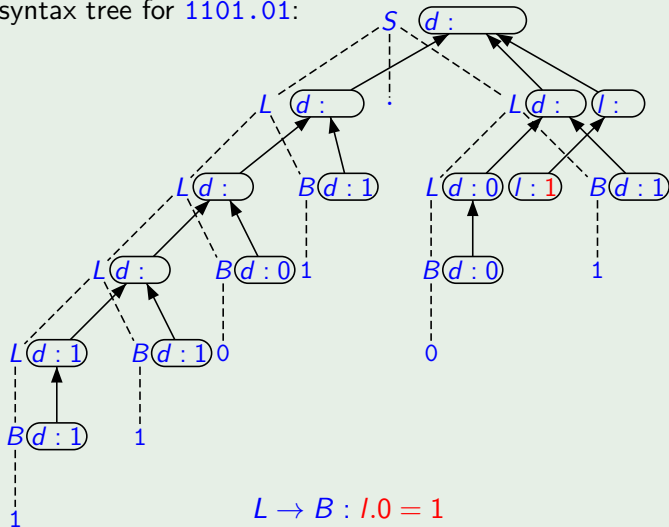
Attributed syntax tree for 1101.01:



Example: Knuth's Binary Numbers II

Example 12.1 (continued)

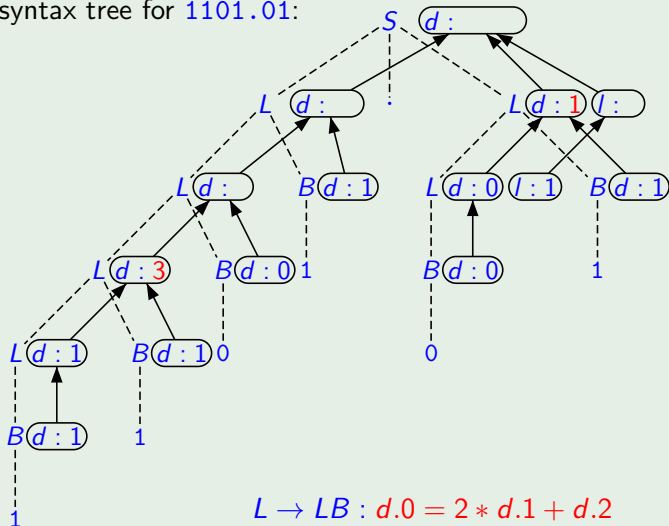
Attributed syntax tree for 1101.01:



Example: Knuth's Binary Numbers II

Example 12.1 (continued)

Attributed syntax tree for 1101.01:

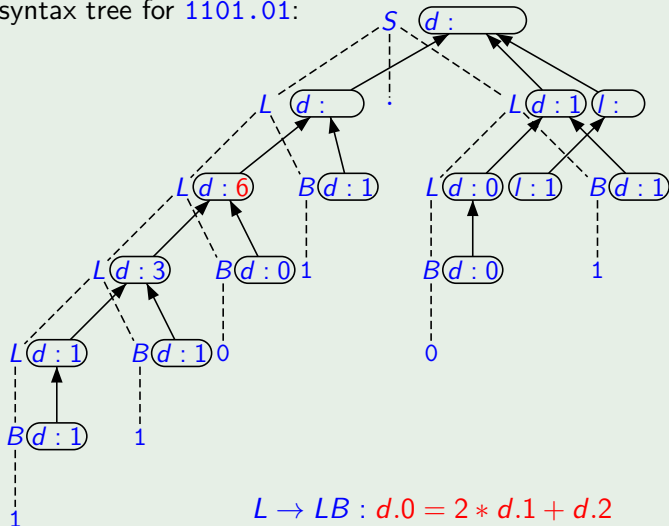


$$L \rightarrow LB : d.0 = 2 * d.1 + d.2$$

Example: Knuth's Binary Numbers II

Example 12.1 (continued)

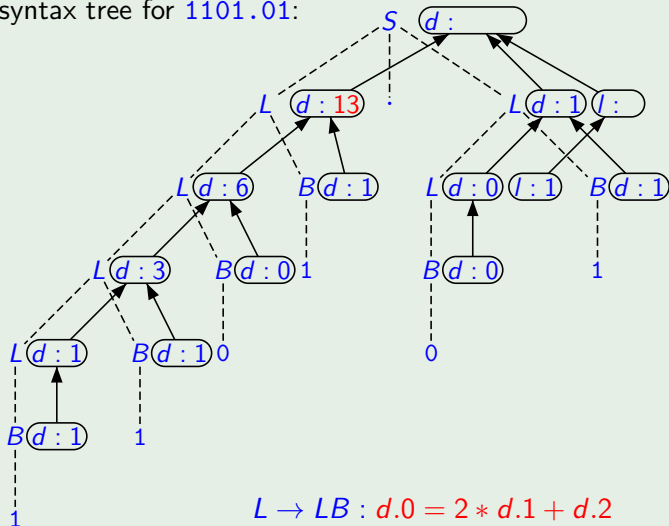
Attributed syntax tree for 1101.01:



Example: Knuth's Binary Numbers II

Example 12.1 (continued)

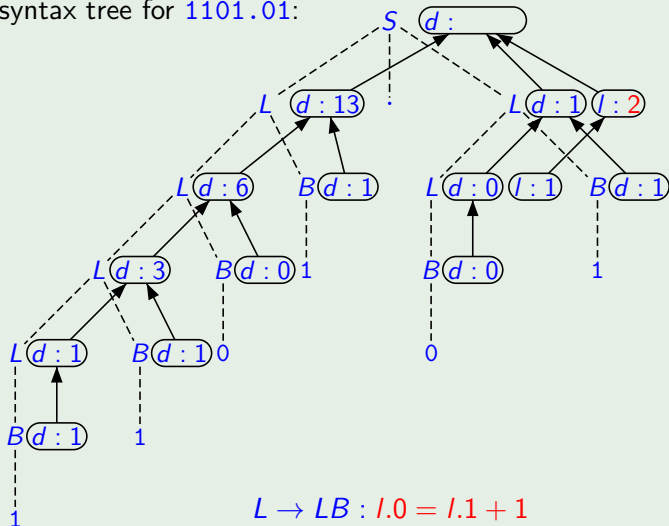
Attributed syntax tree for 1101.01:



Example: Knuth's Binary Numbers II

Example 12.1 (continued)

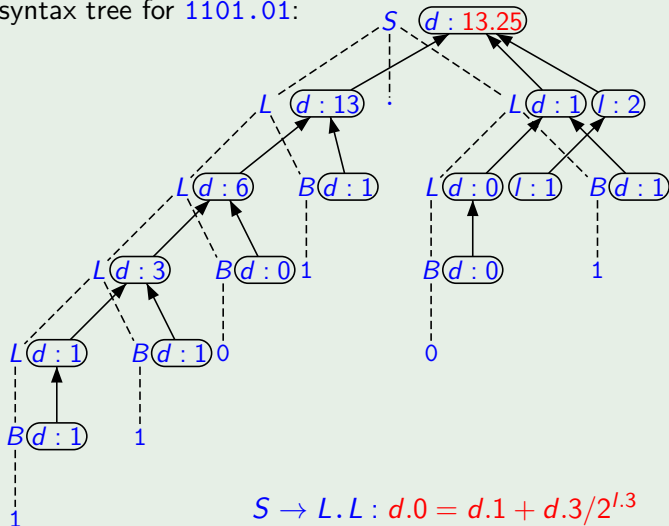
Attributed syntax tree for 1101.01:



Example: Knuth's Binary Numbers II

Example 12.1 (continued)

Attributed syntax tree for 1101.01:



- 1 Overview
- 2 Semantic Analysis
- 3 Attribute Grammars
- 4 Adding Inherited Attributes**
- 5 Formal Definition of Attribute Grammars
- 6 The Attribute Equation System

Adding Inherited Attributes I

Example 12.2 (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 $B \rightarrow 1$

Adding Inherited Attributes I

Example 12.2 (synthesized + inherited attributes)

Binary numbers (with fraction):

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

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

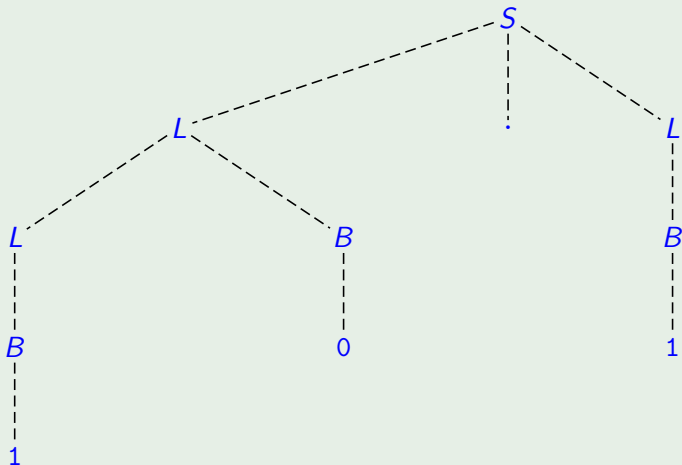
of L : l (length; domain: $V^l := \mathbb{N}$)

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

Adding Inherited Attributes II

Example 12.2 (continued)

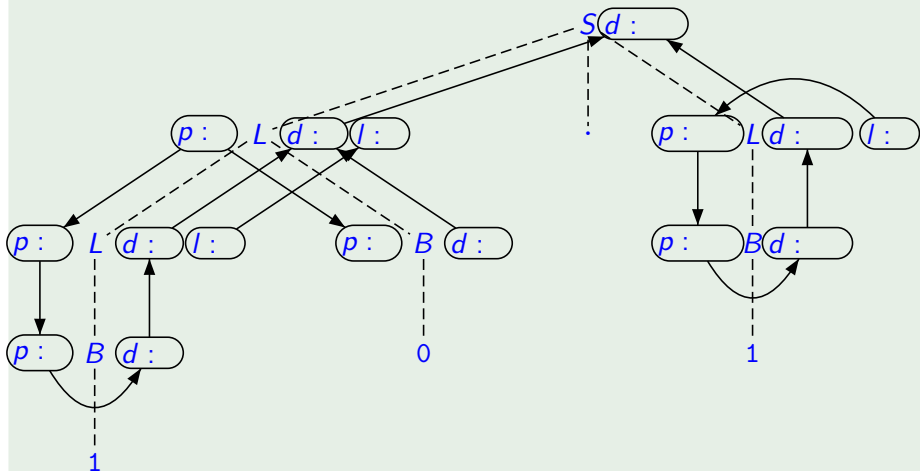
Syntax tree for 10.1:



Adding Inherited Attributes II

Example 12.2 (continued)

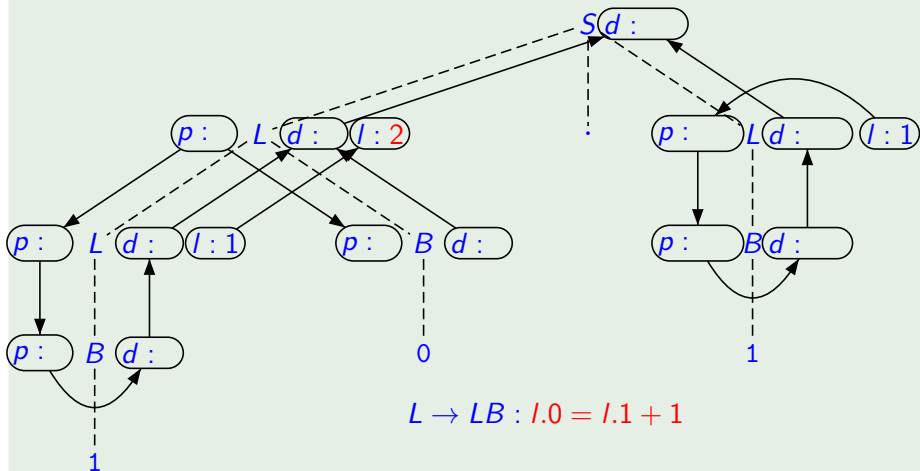
Attributed syntax tree for 10.1:



Adding Inherited Attributes II

Example 12.2 (continued)

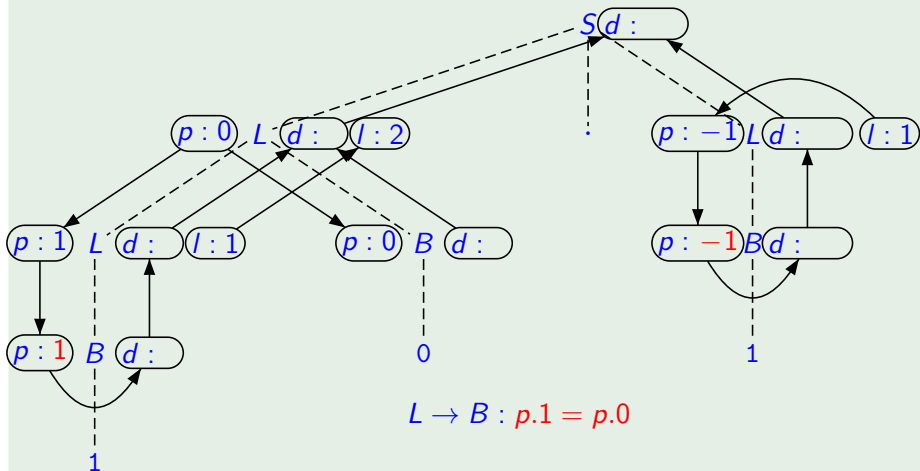
Attributed syntax tree for 10.1:



Adding Inherited Attributes II

Example 12.2 (continued)

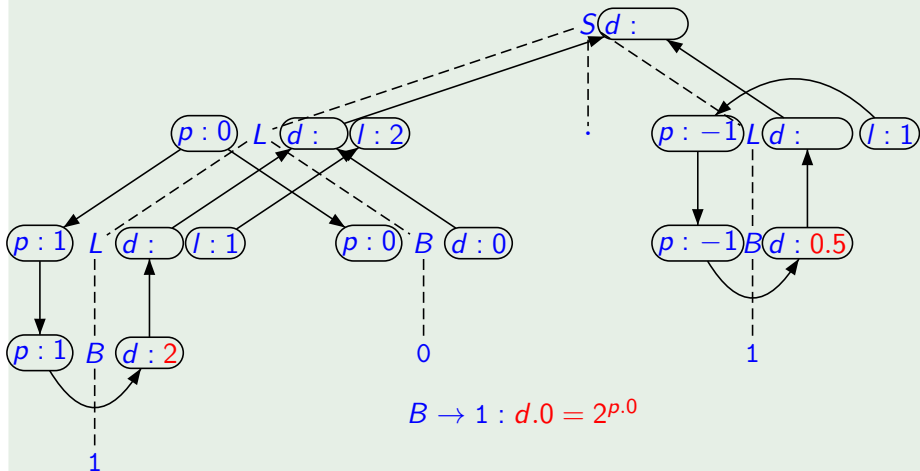
Attributed syntax tree for 10.1:



Adding Inherited Attributes II

Example 12.2 (continued)

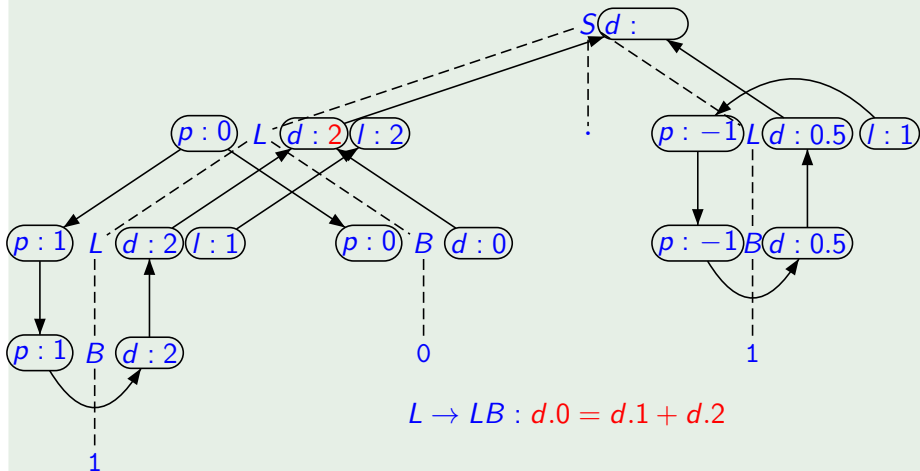
Attributed syntax tree for 10.1:



Adding Inherited Attributes II

Example 12.2 (continued)

Attributed syntax tree for 10.1:



- 1 Overview
- 2 Semantic Analysis
- 3 Attribute Grammars
- 4 Adding Inherited Attributes
- 5 Formal Definition of Attribute Grammars**
- 6 The Attribute Equation System

Definition 12.3 (Attribute grammar)

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

Definition 12.3 (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.

Definition 12.3 (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 $Y \in X$.

Definition 12.3 (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:

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

Definition 12.3 (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 $f : V^{\alpha_1} \times \dots \times V^{\alpha_n} \rightarrow V^{\alpha}$.

Definition 12.3 (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 $E := (E_{\pi} \mid \pi \in P)$.

Formal Definition of Attribute Grammars I

Definition 12.3 (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: $\mathfrak{A} \in AG$.

Example 12.4 (cf. Example 12.2)

$\mathfrak{A}_B \in AG$ for binary numbers:

- **Attributes:** $Att = Syn \uplus Inh$ with $Syn = \{d, l\}$ and $Inh = \{p\}$

Example 12.4 (cf. Example 12.2)

$\mathfrak{A}_B \in AG$ for binary numbers:

- **Attributes:** $Att = Syn \uplus Inh$ with $Syn = \{d, l\}$ and $Inh = \{p\}$
- **Value sets:** $V^d = \mathbb{Q}$, $V^l = \mathbb{N}$, $V^p = \mathbb{Z}$

Example 12.4 (cf. Example 12.2)

$\mathfrak{A}_B \in AG$ for binary numbers:

- **Attributes:** $Att = Syn \uplus Inh$ with $Syn = \{d, l\}$ and $Inh = \{p\}$
- **Value sets:** $V^d = \mathbb{Q}$, $V^l = \mathbb{N}$, $V^p = \mathbb{Z}$
- **Attribute assignment:**

$Y \in X$	S	L	B	0	1	$.$
$syn(Y)$	$\{d\}$	$\{d, l\}$	$\{d\}$	\emptyset	\emptyset	\emptyset
$inh(Y)$	\emptyset	$\{p\}$	$\{p\}$	\emptyset	\emptyset	\emptyset

Formal Definition of Attribute Grammars II

Example 12.4 (cf. Example 12.2)

$\mathfrak{A}_B \in AG$ for binary numbers:

- **Attributes:** $Att = Syn \uplus Inh$ with $Syn = \{d, l\}$ and $Inh = \{p\}$
- **Value sets:** $V^d = \mathbb{Q}$, $V^l = \mathbb{N}$, $V^p = \mathbb{Z}$

- **Attribute assignment:**

$Y \in X$	S	L	B	0	1	$.$
$syn(Y)$	$\{d\}$	$\{d, l\}$	$\{d\}$	\emptyset	\emptyset	\emptyset
$inh(Y)$	\emptyset	$\{p\}$	$\{p\}$	\emptyset	\emptyset	\emptyset

- **Attribute variables:**

$\pi \in P$	$S \rightarrow L$	$S \rightarrow L.L$	$L \rightarrow B$
In_π	$\{d.0, p.1\}$	$\{d.0, p.1, p.3\}$	$\{d.0, l.0, p.1\}$
Out_π	$\{d.1, l.1\}$	$\{d.1, l.1, d.3, l.3\}$	$\{d.1, p.0\}$

$\pi \in P$	$L \rightarrow LB$	$B \rightarrow 0$	$B \rightarrow 1$
In_π	$\{d.0, l.0, p.1, p.2\}$	$\{d.0\}$	$\{d.0\}$
Out_π	$\{d.1, d.2, l.1, p.0\}$	$\{p.0\}$	$\{p.0\}$

Formal Definition of Attribute Grammars II

Example 12.4 (cf. Example 12.2)

$\mathfrak{A}_B \in AG$ for binary numbers:

- **Attributes:** $Att = Syn \uplus Inh$ with $Syn = \{d, l\}$ and $Inh = \{p\}$
- **Value sets:** $V^d = \mathbb{Q}$, $V^l = \mathbb{N}$, $V^p = \mathbb{Z}$

- **Attribute assignment:**

$Y \in X$	S	L	B	0	1	$.$
$syn(Y)$	$\{d\}$	$\{d, l\}$	$\{d\}$	\emptyset	\emptyset	\emptyset
$inh(Y)$	\emptyset	$\{p\}$	$\{p\}$	\emptyset	\emptyset	\emptyset

- **Attribute variables:**

$\pi \in P$	$S \rightarrow L$	$S \rightarrow L.L$	$L \rightarrow B$
In_π	$\{d.0, p.1\}$	$\{d.0, p.1, p.3\}$	$\{d.0, l.0, p.1\}$
Out_π	$\{d.1, l.1\}$	$\{d.1, l.1, d.3, l.3\}$	$\{d.1, p.0\}$

$\pi \in P$	$L \rightarrow LB$	$B \rightarrow 0$	$B \rightarrow 1$
In_π	$\{d.0, l.0, p.1, p.2\}$	$\{d.0\}$	$\{d.0\}$
Out_π	$\{d.1, d.2, l.1, p.0\}$	$\{p.0\}$	$\{p.0\}$

- **Semantic rules:** see Example 12.2
(e.g., $E_{S \rightarrow L} = \{d.0 = d.1, p.1 = 0\}$)

- 1 Overview
- 2 Semantic Analysis
- 3 Attribute Grammars
- 4 Adding Inherited Attributes
- 5 Formal Definition of Attribute Grammars
- 6 The Attribute Equation System**

Definition 12.5 (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** :

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

Definition 12.5 (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 .

Definition 12.5 (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** :

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

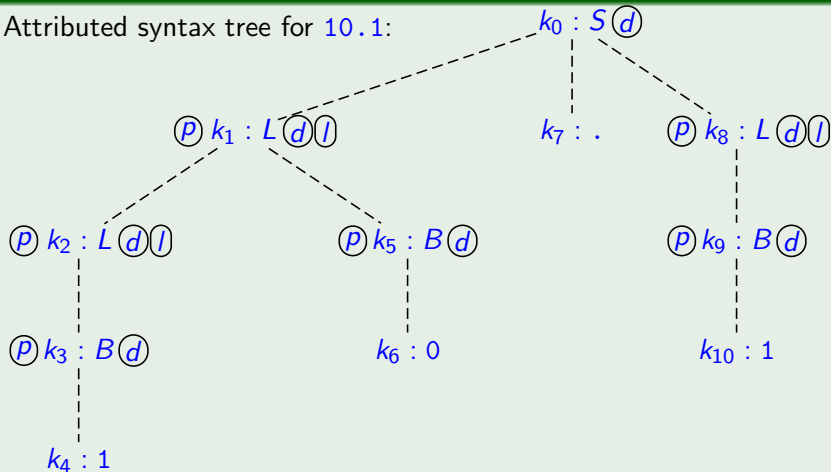
- Let $k_0 \in K$ be an (inner) node where production $\pi = Y_0 \rightarrow Y_1 \dots Y_r \in P$ is applied, and let $k_1, \dots, k_r \in K$ be the corresponding successor nodes. The **attribute equation system E_{k_0}** of k_0 is obtained from E_π by substituting every attribute index $i \in \{0, \dots,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

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

Attribution of Syntax Trees II

Example 12.6 (cf. Example 12.2)

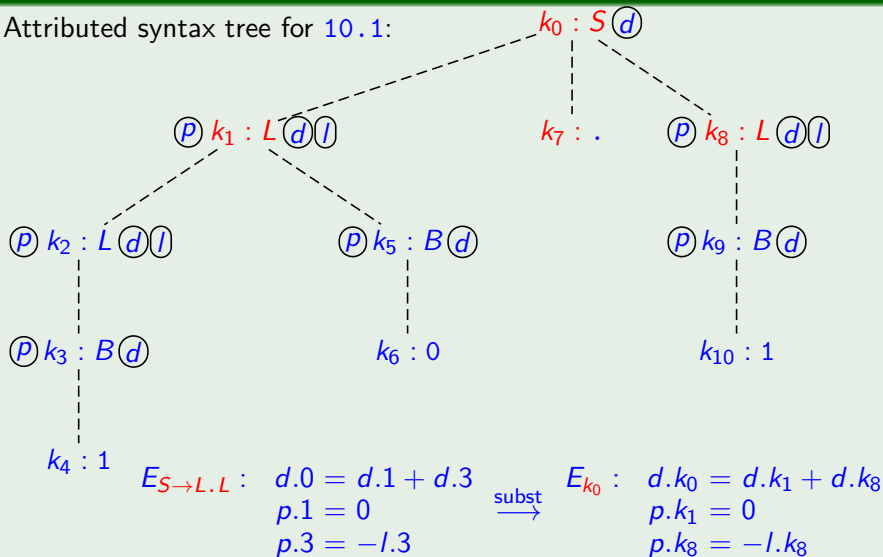
Attributed syntax tree for 10.1:



Attribution of Syntax Trees II

Example 12.6 (cf. Example 12.2)

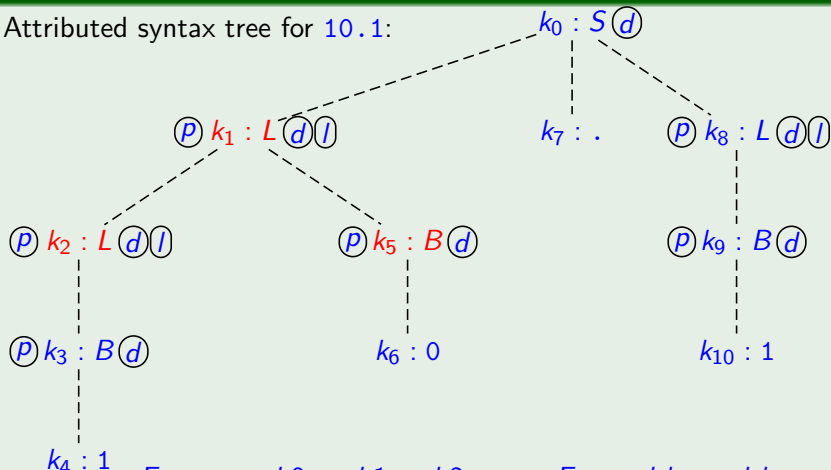
Attributed syntax tree for 10.1:



Attribution of Syntax Trees II

Example 12.6 (cf. Example 12.2)

Attributed syntax tree for 10.1:



$k_4 : 1$

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

subst
→

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

Corollary 12.7

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 12.7

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.