

Compiler Construction

Lecture 13: Semantic Analysis I (Definition of Attribute Grammars)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH Aachen University

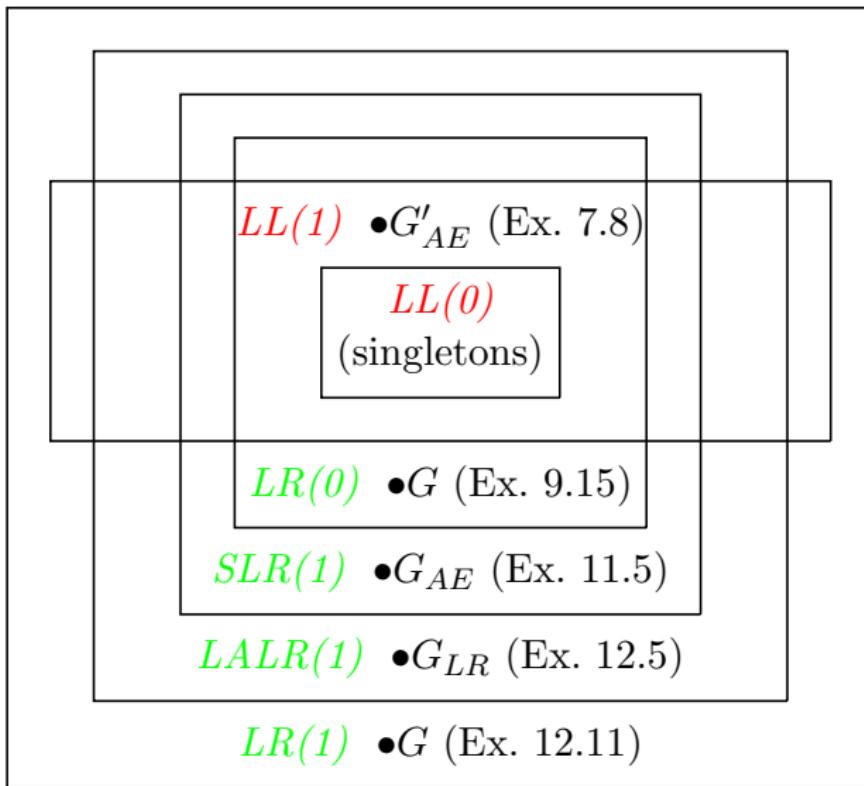
noll@cs.rwth-aachen.de

<http://www-i2.informatik.rwth-aachen.de/i2/cc08/>

Summer semester 2008

- 1 Repetition: Expressiveness of LL and LR Grammars
- 2 LL and LR Parsing in Practice
- 3 Overview
- 4 Problem Statement
- 5 Attribute Grammars
- 6 Formal Definition of Attribute Grammars

Overview of Grammar Classes

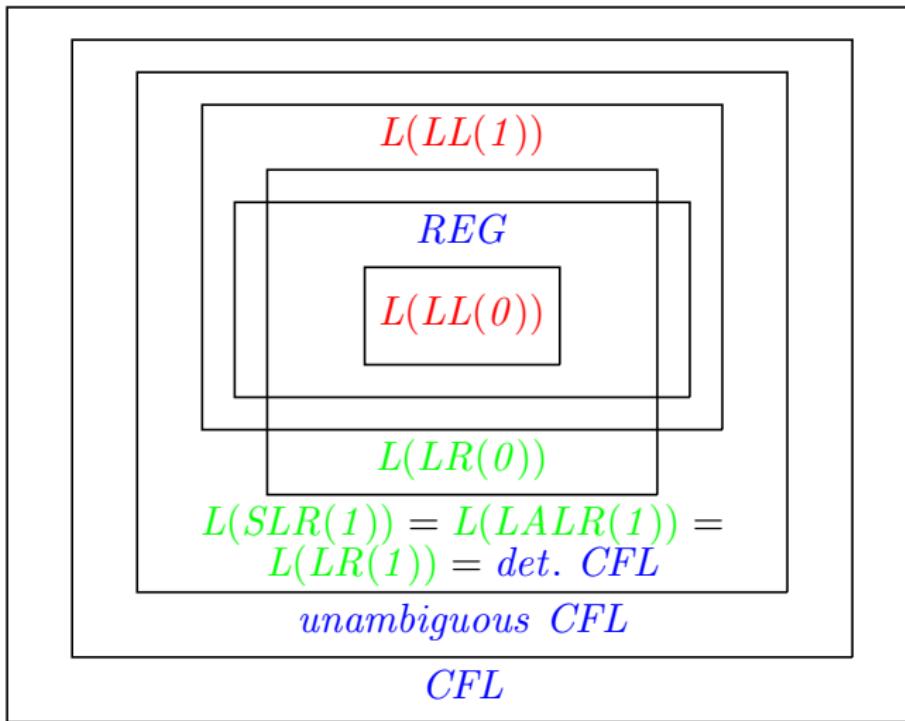


Moreover:

- $LL(k) \subsetneq LL(k+1)$
for every $k \in \mathbb{N}$
- $LR(k) \subsetneq LR(k+1)$
for every $k \in \mathbb{N}$
- $LL(k) \subseteq LR(k)$
for every $k \in \mathbb{N}$

Overview of Language Classes

(cf. O. Mayer: Syntaxanalyse, BI-Verlag, 1978, p. 409ff)



Moreover:

- $L(LL(k)) \subsetneq L(LL(k+1)) \subsetneq L(LR(1))$
for every $k \in \mathbb{N}$
- $L(LR(k)) = L(LR(1))$
for every $k \geq 1$

- 1 Repetition: Expressiveness of LL and LR Grammars
- 2 LL and LR Parsing in Practice
- 3 Overview
- 4 Problem Statement
- 5 Attribute Grammars
- 6 Formal Definition of Attribute Grammars

LL and LR Parsing in Practice

In practice: use of *LL(1)* or *LALR(1)*

Detailed comparison (cf. Fischer/LeBlanc: *Crafting a Compiler*, Benjamin/Cummings, 1988):

Simplicity : LL wins

- LL parsing technique easier to understand
- recursive-descent parser easier to debug than LALR action tables

Generality : LALR wins

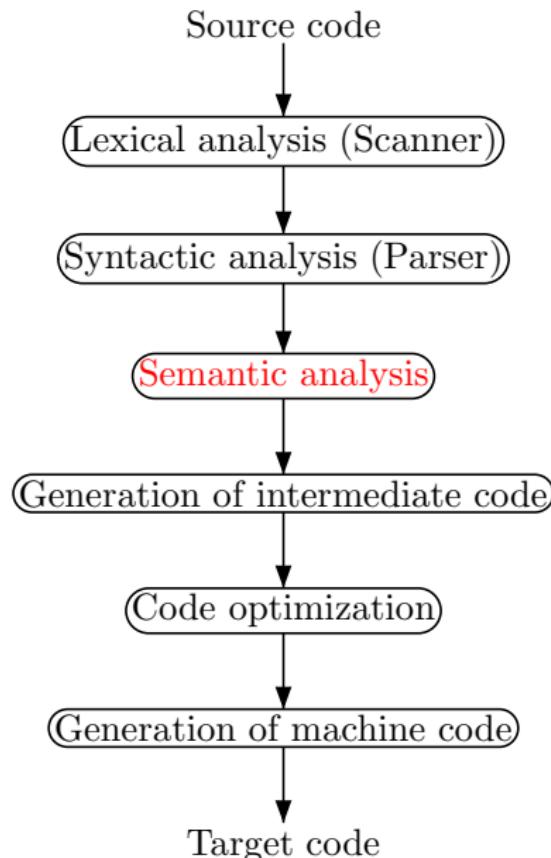
- “almost” $LL(1) \subseteq LALR(1)$ (only pathological counterexamples)
- LL requires elimination of left recursion and left factorization

Semantic actions : (see semantic analysis) LL wins

- actions can be placed anywhere in LL parsers without causing conflicts
- in LALR: implicit ε -productions

- 1 Repetition: Expressiveness of LL and LR Grammars
- 2 LL and LR Parsing in Practice
- 3 Overview
- 4 Problem Statement
- 5 Attribute Grammars
- 6 Formal Definition of Attribute Grammars

Conceptual Structure of a Compiler



- 1 Repetition: Expressiveness of LL and LR Grammars
- 2 LL and LR Parsing in Practice
- 3 Overview
- 4 Problem Statement
- 5 Attribute Grammars
- 6 Formal Definition of Attribute 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 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, ...

These properties are determined by

Scope rules: defines part of program where a declaration is **valid**

Visibility rules: defines part of scope where a declaration is **visible**
(overlapping of global and local declarations)

Typing rules: defines **type consistency** of expressions, statements, ...

- 1 Repetition: Expressiveness of LL and LR Grammars
- 2 LL and LR Parsing in Practice
- 3 Overview
- 4 Problem Statement
- 5 Attribute Grammars
- 6 Formal Definition of Attribute Grammars

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 leafs to the root)
 - Inherited:** top-down computation (from the root to the leafs)
- 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: Knuth's Binary Numbers I

Example 13.1 (only synthesized attributes)

Binary numbers (with fraction):

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

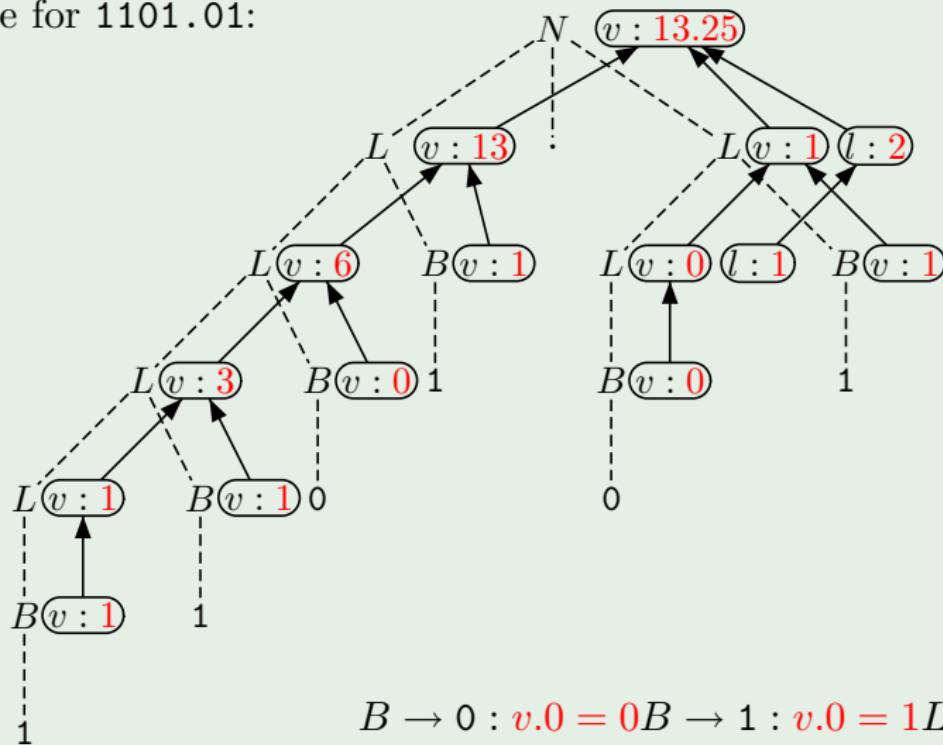
Synthesized attributes of N, L, B : v (value; domain: $V^v := \mathbb{Q}$)
of L : l (length; domain: $V^l := \mathbb{N}$)

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

Example: Knuth's Binary Numbers II

Example 13.1 (continued)

Syntax tree for 1101.01:



$v.0 = v.1 L \rightarrow B : l.0 = 1 L \rightarrow LB : v.0 = 2 * v.1 + v.2 L \rightarrow LB : l.0 =$

$B \rightarrow 0 : v.0 = 0 B \rightarrow 1 : v.0 = 1 L \rightarrow B :$

Adding Inherited Attributes I

Example 13.2 (synthesized + inherited attributes)

Binary numbers (with fraction):

G'_B : Numbers	$N \rightarrow L$	$v.0 = v.1$
		$p.1 = 0$
	$N \rightarrow L.L$	$v.0 = v.1 + v.3$
		$p.1 = 0$
		$p.3 = -l.3$
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 N, L, B : v (value; domain: $V^v := \mathbb{Q}$)

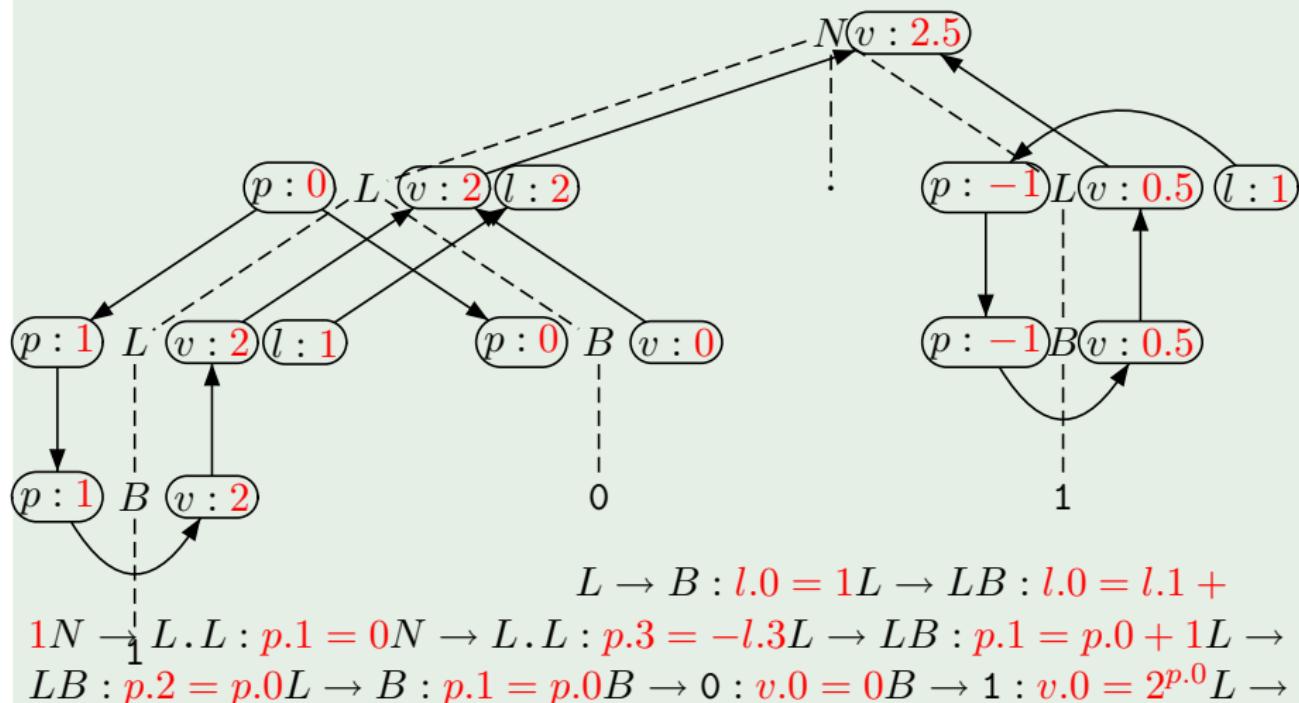
of L : l (length; domain: $V^l := \mathbb{N}$)

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

Adding Inherited Attributes II

Example 13.2 (continued)

Syntax tree for 10.1:



- 1 Repetition: Expressiveness of LL and LR Grammars
- 2 LL and LR Parsing in Practice
- 3 Overview
- 4 Problem Statement
- 5 Attribute Grammars
- 6 Formal Definition of Attribute Grammars

Definition 13.3 (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 13.4 (cf. Example 13.2)

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

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

$Y \in X$	N	L	B	0	1
$syn(Y)$	$\{v\}$	$\{v, l\}$	$\{v\}$	\emptyset	\emptyset
$inh(Y)$	\emptyset	$\{p\}$	$\{p\}$	\emptyset	\emptyset

- Attribute variables:

$\pi \in P$	$N \rightarrow L$	$N \rightarrow L.L$	$L \rightarrow B$
In_π	$\{v.0, p.1\}$	$\{v.0, p.1, p.3\}$	$\{v.0, l.0, p.1\}$
Out_π	$\{v.1, l.1\}$	$\{v.1, l.1, v.3, l.3\}$	$\{v.1, p.0\}$
$\pi \in P$	$L \rightarrow LB$	$B \rightarrow 0$	$B \rightarrow 1$
In_π	$\{v.0, l.0, p.1, p.2\}$	$\{v.0\}$	$\{v.0\}$
Out_π	$\{v.1, v.2, l.1, p.0\}$	$\{p.0\}$	$\{p.0\}$

- Semantic rules: see Example 13.2
(e.g., $E_{N \rightarrow L} = \{v.0 = v.1, p.1 = 0\}$)

Definition 13.5 (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

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

Corollary 13.6

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

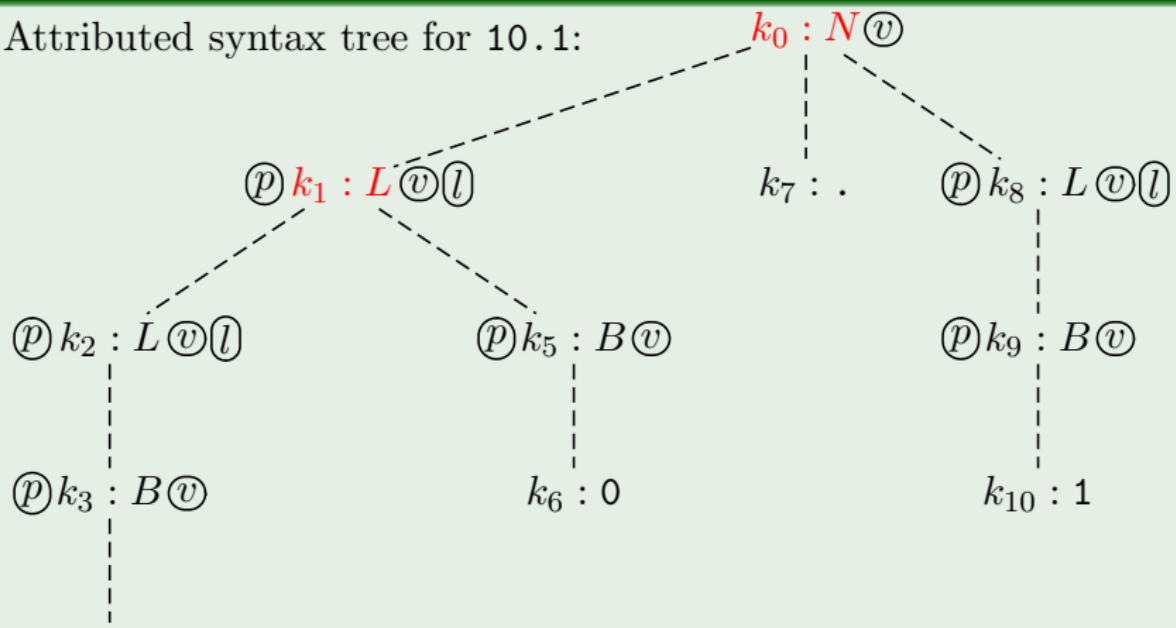
Assumptions:

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

Attribution of Syntax Trees III

Example 13.7 (cf. Example 13.2)

Attributed syntax tree for 10.1:



$$E_{N \rightarrow L.L} : \begin{array}{l} v.0 = v.1 + v.3 \\ p.1 = 0 \\ p.3 = -l.3 \end{array} \xrightarrow{\text{subst}} E_{k_0} : \begin{array}{l} v.k_0 = v.k_1 + v.k_8 \\ p.k_1 = 0 \\ p.k_8 = -l.k_8 \end{array}$$

$$E_{L \rightarrow L.R} : v.0 = v.1 + v.2$$

$$E_{k_1} : v.k_1 = v.k_2 + v.k_5$$