

Compiler Construction

Lecture 15: Semantic Analysis I (Attribute Grammars)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

RWTH Aachen University

`noll@cs.rwth-aachen.de`

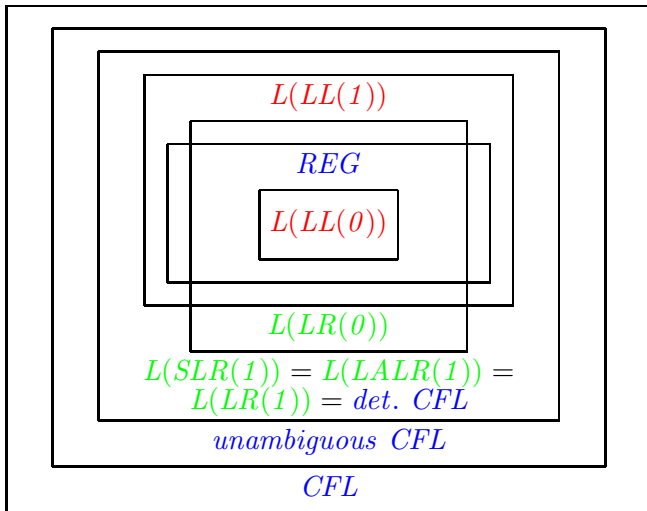
`http://www-i2.informatik.rwth-aachen.de/i2/cc10/`

Winter semester 2010/11

- 1 Regular vs. $LR(0)$ Languages
- 2 Overview
- 3 Semantic Analysis
- 4 Attribute Grammars
- 5 Adding Inherited Attributes
- 6 Formal Definition of Attribute Grammars

Repetition: 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 $k \geq 1$

Why $REG \not\subseteq L(LR(0))$?

Definition (cf. Exercise 6.2)

A language $L \subseteq \Sigma^*$ is called **prefix-free** if $L \cap L \cdot \Sigma^+ = \emptyset$, i.e., if no proper prefix of an element of L is again in L .

Why $REG \not\subseteq L(LR(0))$?

Definition (cf. Exercise 6.2)

A language $L \subseteq \Sigma^*$ is called **prefix-free** if $L \cap L \cdot \Sigma^+ = \emptyset$, i.e., if no proper prefix of an element of L is again in L .

Lemma (cf. Exercise 6.2)

$G \in LR(0) \implies L(G)$ *prefix-free*

Why $REG \not\subseteq L(LR(0))$?

Definition (cf. Exercise 6.2)

A language $L \subseteq \Sigma^*$ is called **prefix-free** if $L \cap L \cdot \Sigma^+ = \emptyset$, i.e., if no proper prefix of an element of L is again in L .

Lemma (cf. Exercise 6.2)

$G \in LR(0) \implies L(G) \text{ prefix-free}$

Corollary

$\{a, aa\} \in REG \setminus L(LR(0))$

Why $REG \not\subseteq L(LR(0))$?

Definition (cf. Exercise 6.2)

A language $L \subseteq \Sigma^*$ is called **prefix-free** if $L \cap L \cdot \Sigma^+ = \emptyset$, i.e., if no proper prefix of an element of L is again in L .

Lemma (cf. Exercise 6.2)

$G \in LR(0) \implies L(G)$ *prefix-free*

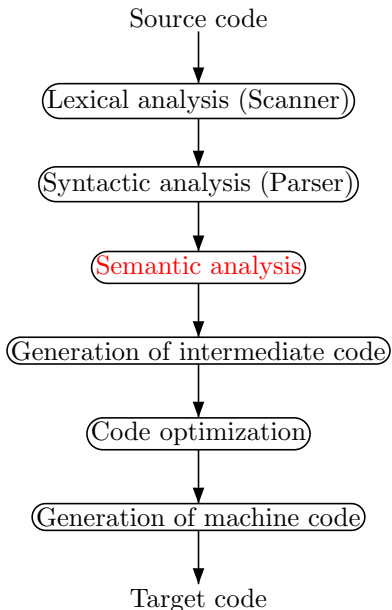
Corollary

$\{a, aa\} \in REG \setminus L(LR(0))$

Conjecture: $L \in REG \setminus L(LR(0)) \implies L(G)$ not prefix-free?
(cf. Exercise 7)

- 1 Regular vs. $LR(0)$ Languages
- 2 Overview
- 3 Semantic Analysis
- 4 Attribute Grammars
- 5 Adding Inherited Attributes
- 6 Formal Definition of Attribute Grammars

Conceptual Structure of a Compiler



- 1 Regular vs. $LR(0)$ Languages
- 2 Overview
- 3 Semantic Analysis
- 4 Attribute Grammars
- 5 Adding Inherited Attributes
- 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?

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 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 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, ...

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, ...

Among others, these properties are determined by

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

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

- 1 Regular vs. $LR(0)$ Languages
- 2 Overview
- 3 Semantic Analysis
- 4 Attribute Grammars
- 5 Adding Inherited Attributes
- 6 Formal Definition of 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**

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

Result: **attributed syntax tree**

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**

$\Rightarrow$ **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: Knuth's Binary Numbers I

Example 15.1 (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 15.1 (only synthesized attributes)

Binary numbers (with fraction):

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

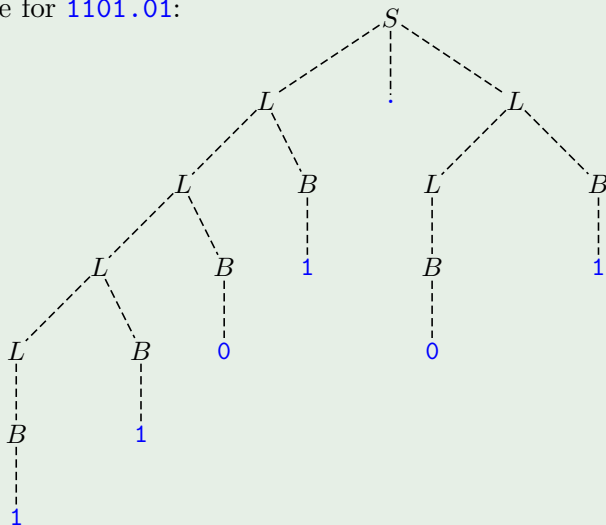
Synthesized attributes of S, L, B : v (value; domain: $V^v := \mathbb{Q}$)
of L : l (length; domain: $V^l := \mathbb{N}$)

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

Example: Knuth's Binary Numbers II

Example 15.1 (continued)

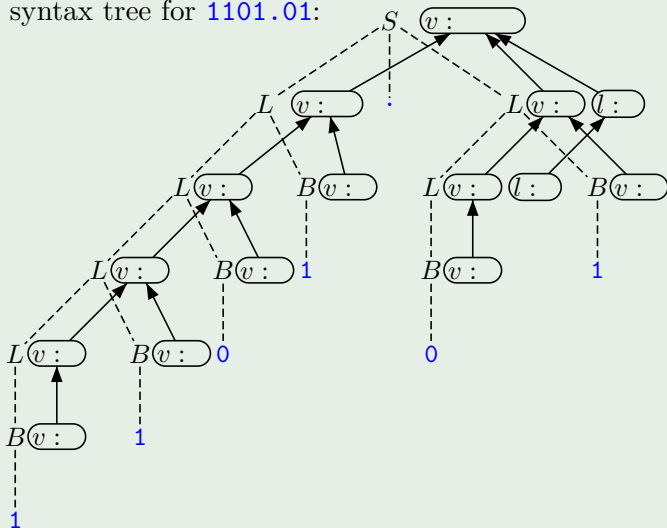
Syntax tree for 1101.01:



Example: Knuth's Binary Numbers II

Example 15.1 (continued)

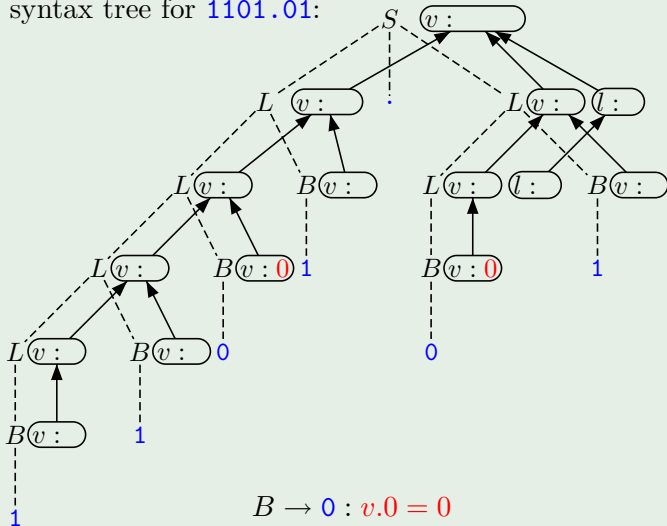
Attributed syntax tree for 1101.01:



Example: Knuth's Binary Numbers II

Example 15.1 (continued)

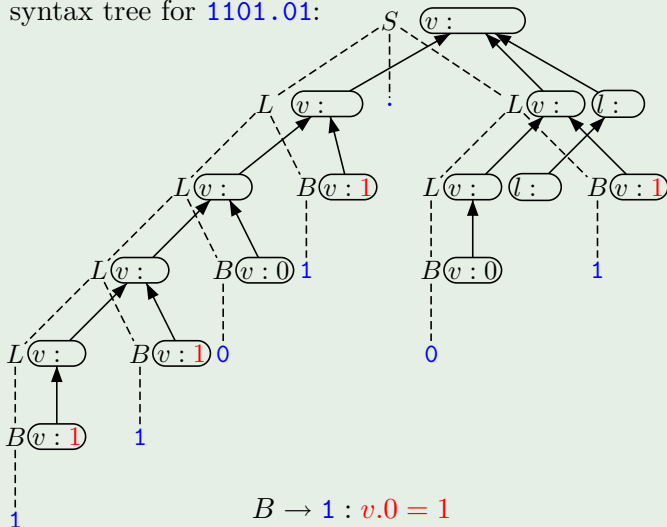
Attributed syntax tree for 1101.01:



Example: Knuth's Binary Numbers II

Example 15.1 (continued)

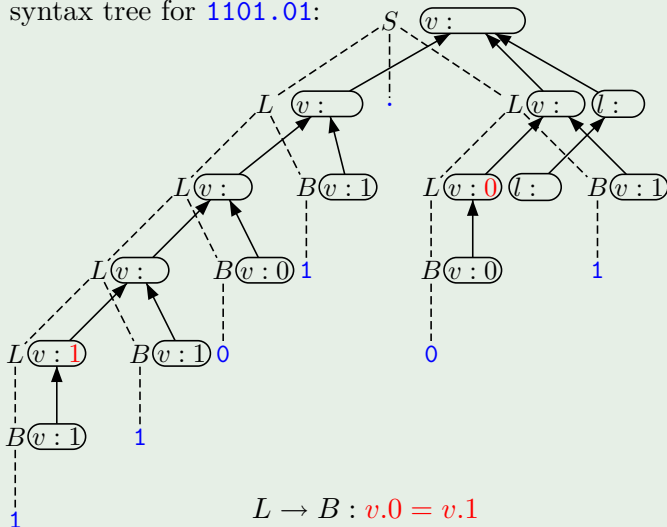
Attributed syntax tree for 1101.01:



Example: Knuth's Binary Numbers II

Example 15.1 (continued)

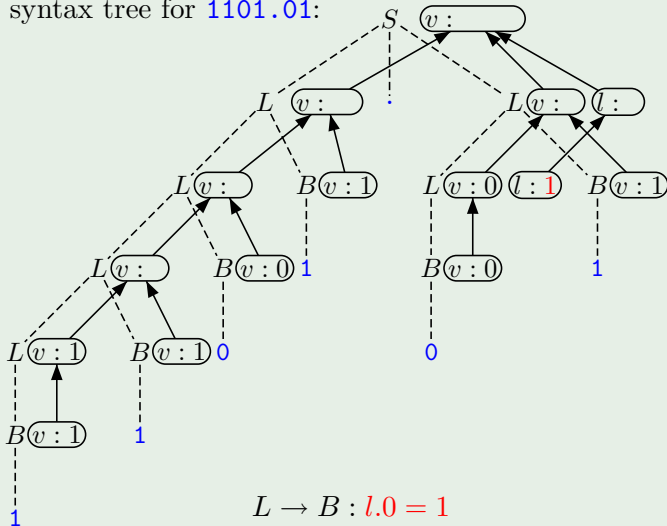
Attributed syntax tree for 1101.01:



Example: Knuth's Binary Numbers II

Example 15.1 (continued)

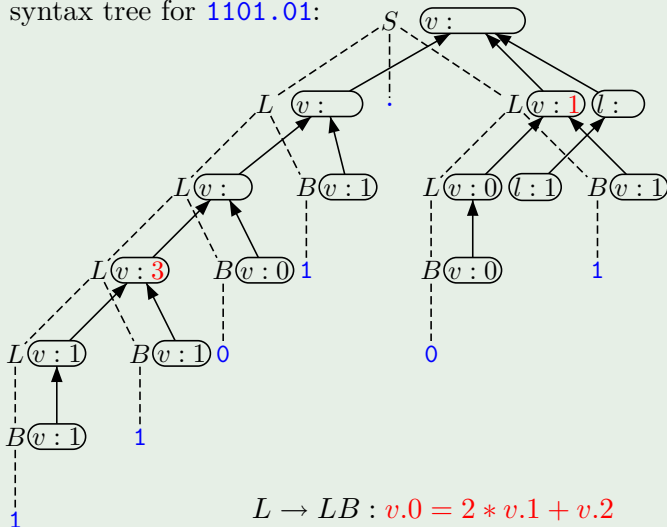
Attributed syntax tree for 1101.01:



Example: Knuth's Binary Numbers II

Example 15.1 (continued)

Attributed syntax tree for 1101.01:

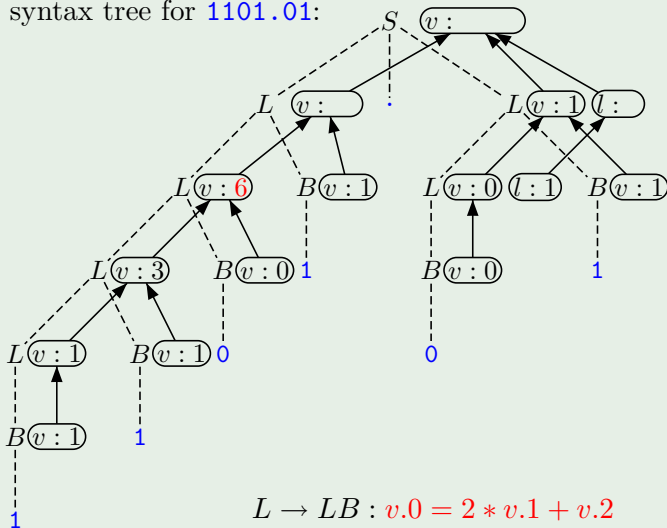


$$L \rightarrow LB : v.0 = 2 * v.1 + v.2$$

Example: Knuth's Binary Numbers II

Example 15.1 (continued)

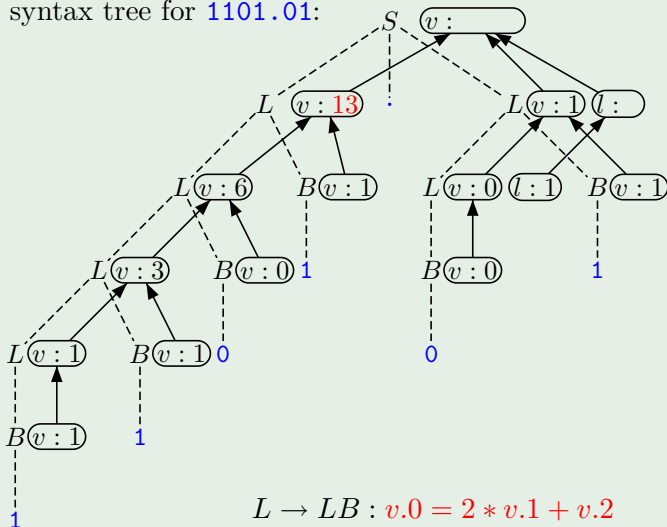
Attributed syntax tree for 1101.01:



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

Example 15.1 (continued)

Attributed syntax tree for 1101.01:

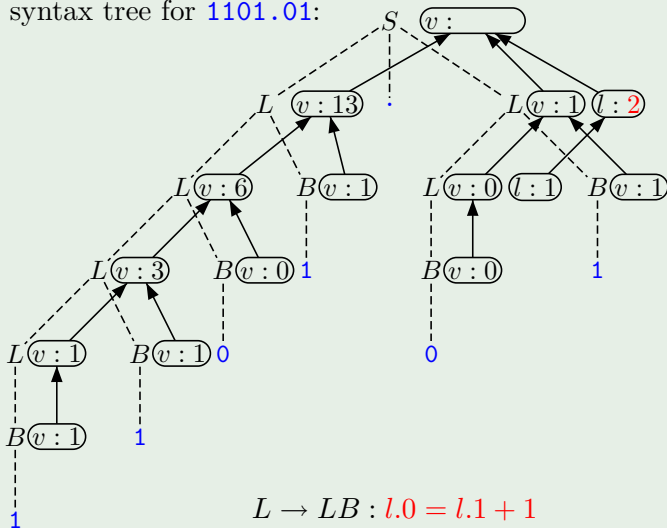


$$L \rightarrow LB : v.0 = 2 * v.1 + v.2$$

Example: Knuth's Binary Numbers II

Example 15.1 (continued)

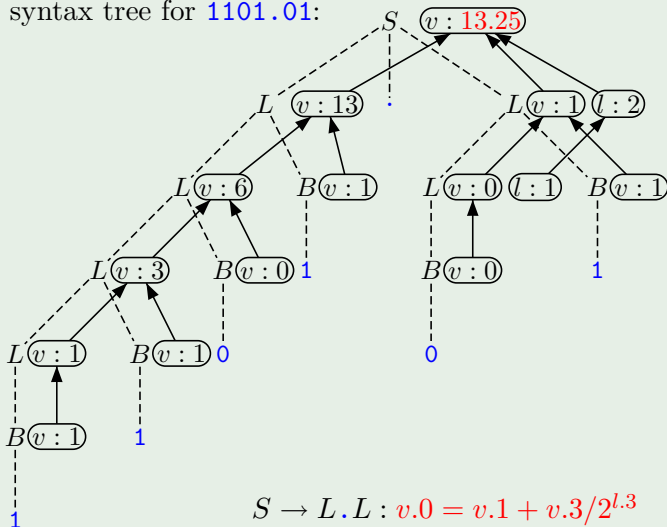
Attributed syntax tree for 1101.01:



Example: Knuth's Binary Numbers II

Example 15.1 (continued)

Attributed syntax tree for 1101.01:



$$S \rightarrow L.L : v.0 = v.1 + v.3/2^{l.3}$$

- 1 Regular vs. $LR(0)$ Languages
- 2 Overview
- 3 Semantic Analysis
- 4 Attribute Grammars
- 5 Adding Inherited Attributes
- 6 Formal Definition of Attribute Grammars

Example 15.2 (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 15.2 (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: $V^v := \mathbb{Q}$)

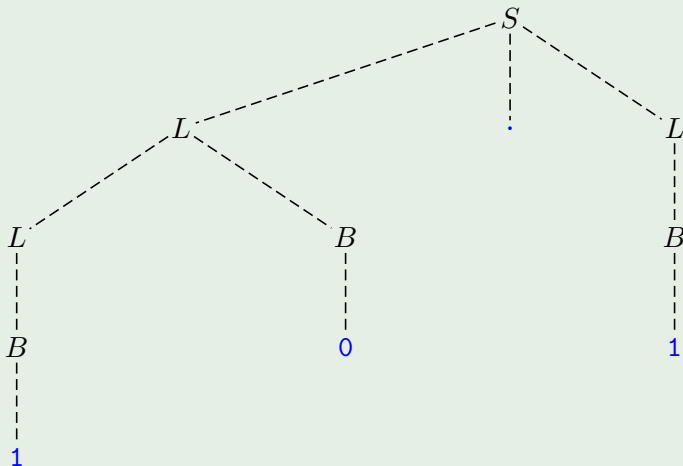
of L : l (length; domain: $V^l := \mathbb{N}$)

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

Adding Inherited Attributes II

Example 15.2 (continued)

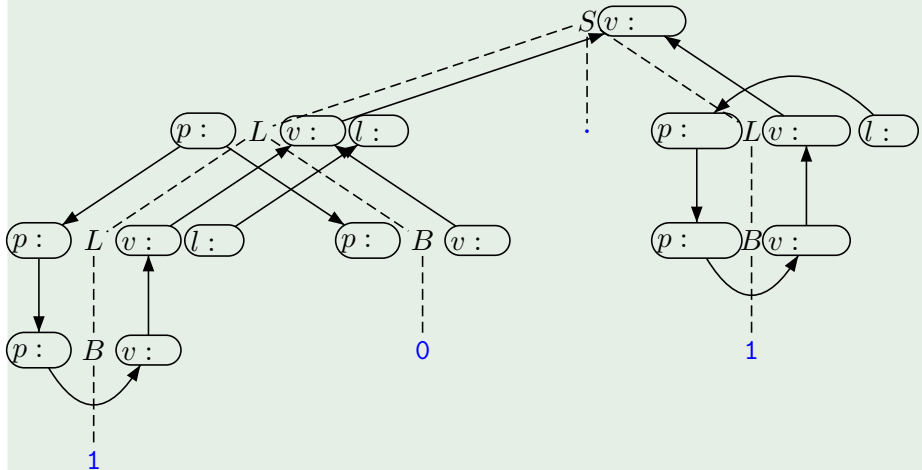
Syntax tree for 10.1:



Adding Inherited Attributes II

Example 15.2 (continued)

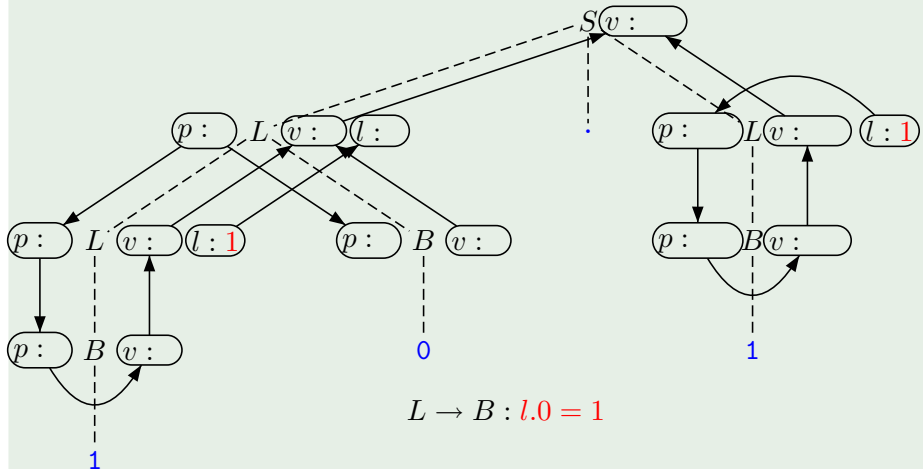
Attributed syntax tree for 10.1:



Adding Inherited Attributes II

Example 15.2 (continued)

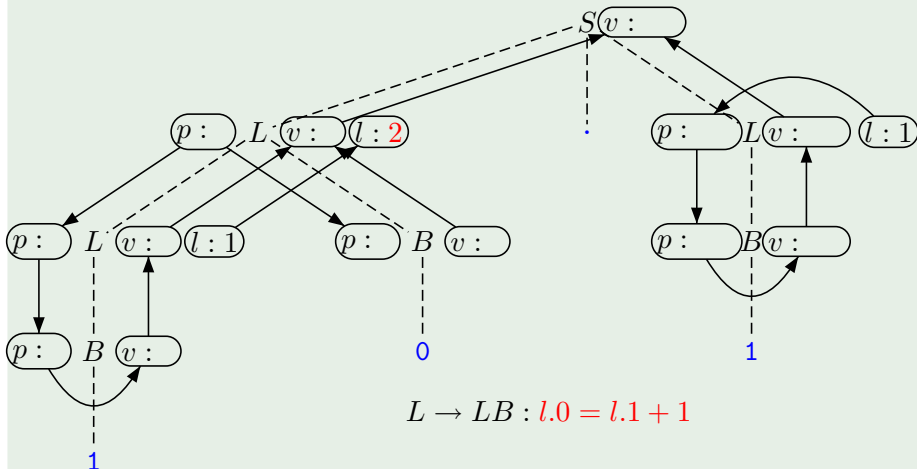
Attributed syntax tree for 10.1:



Adding Inherited Attributes II

Example 15.2 (continued)

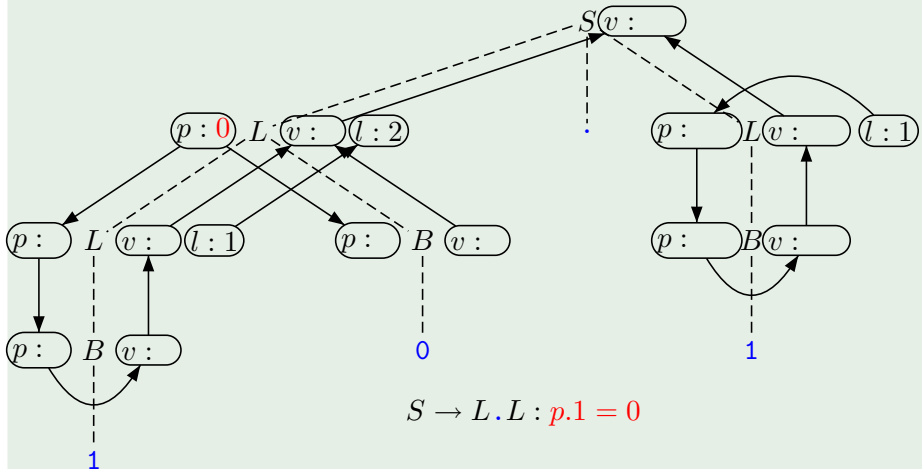
Attributed syntax tree for 10.1:



Adding Inherited Attributes II

Example 15.2 (continued)

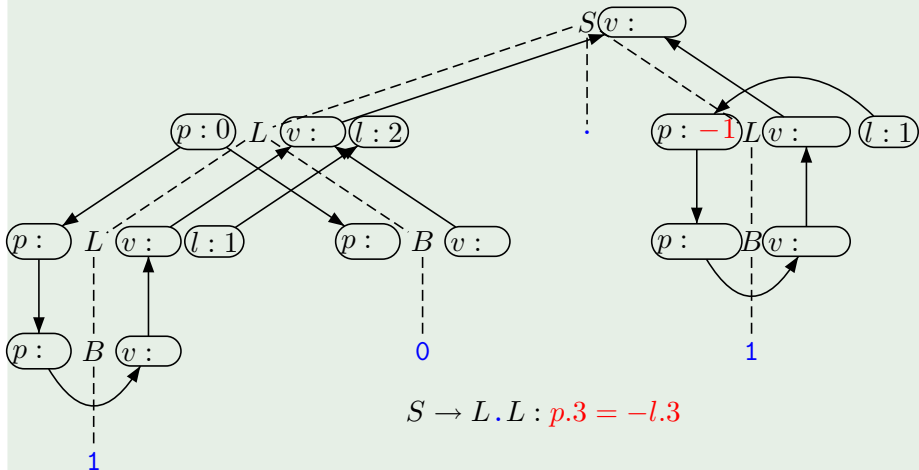
Attributed syntax tree for 10.1:



Adding Inherited Attributes II

Example 15.2 (continued)

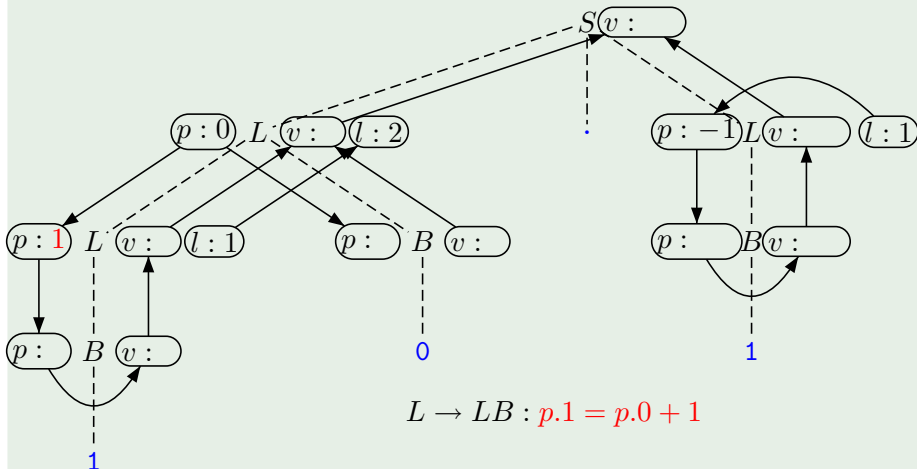
Attributed syntax tree for 10.1:



Adding Inherited Attributes II

Example 15.2 (continued)

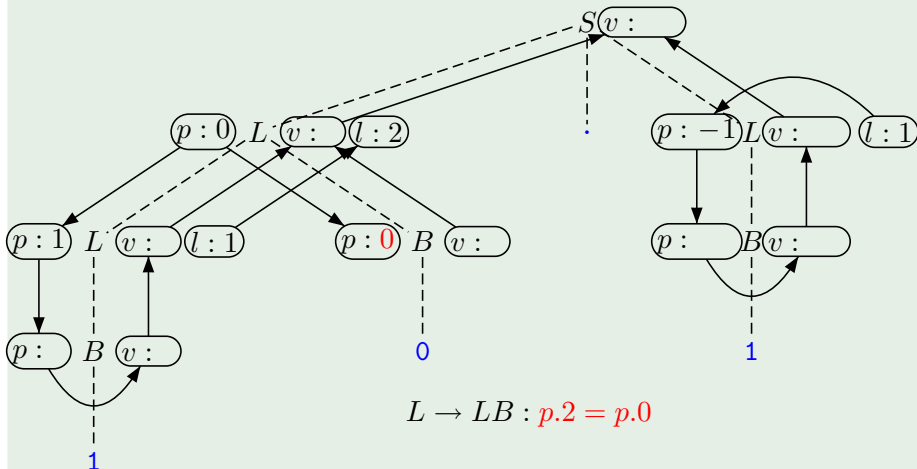
Attributed syntax tree for 10.1:



Adding Inherited Attributes II

Example 15.2 (continued)

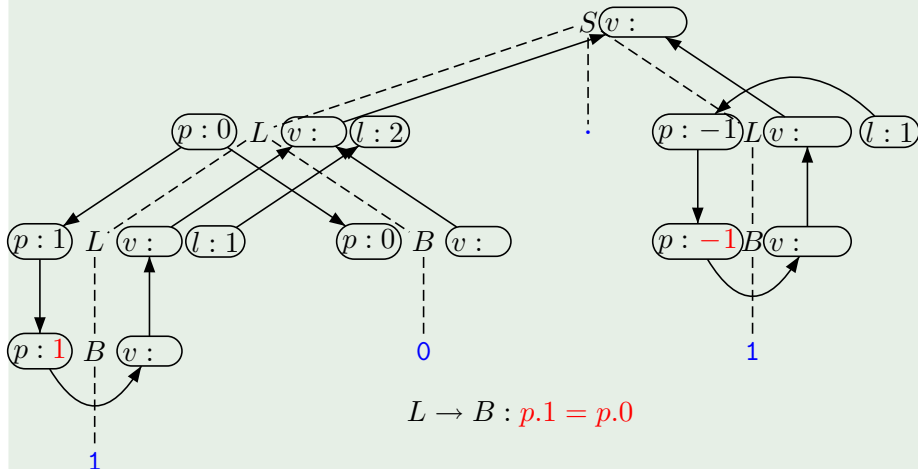
Attributed syntax tree for 10.1:



Adding Inherited Attributes II

Example 15.2 (continued)

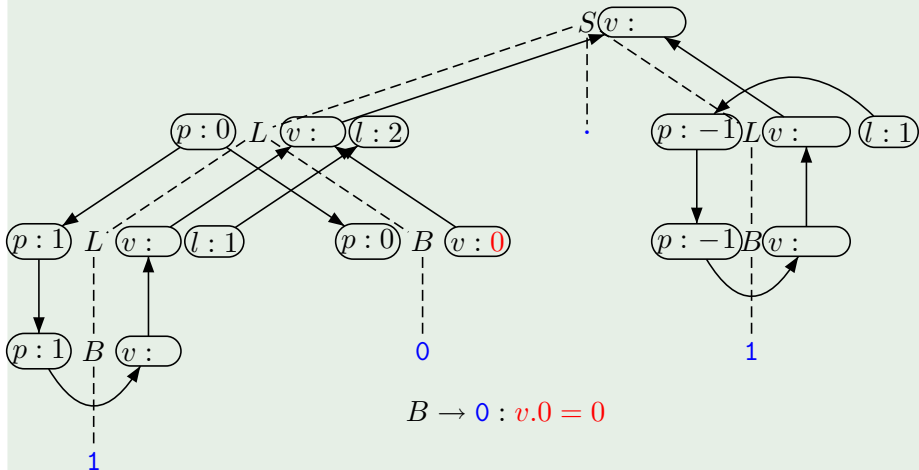
Attributed syntax tree for 10.1:



Adding Inherited Attributes II

Example 15.2 (continued)

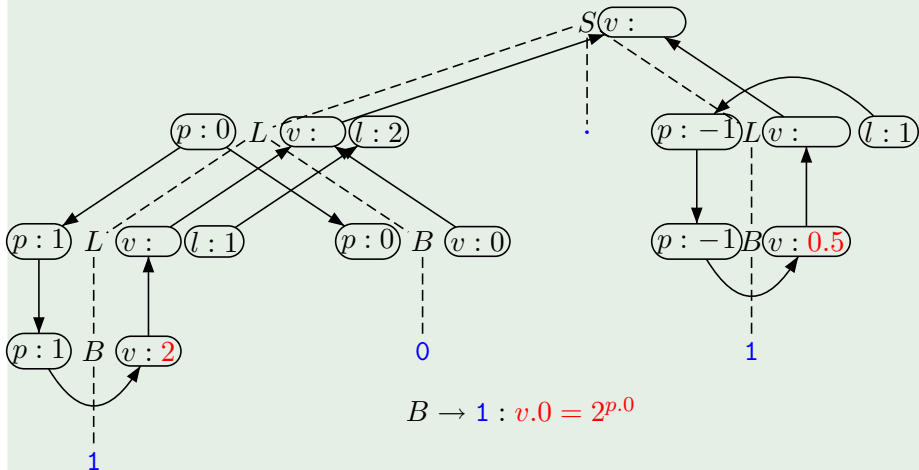
Attributed syntax tree for 10.1:



Adding Inherited Attributes II

Example 15.2 (continued)

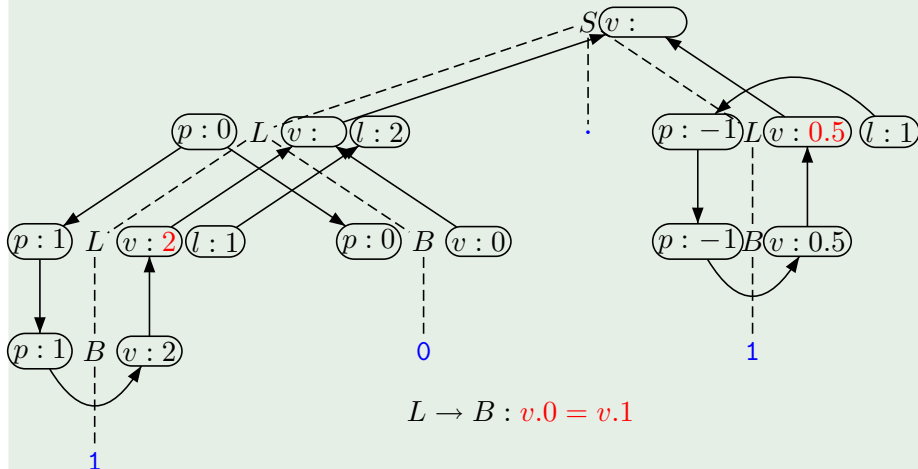
Attributed syntax tree for 10.1:



Adding Inherited Attributes II

Example 15.2 (continued)

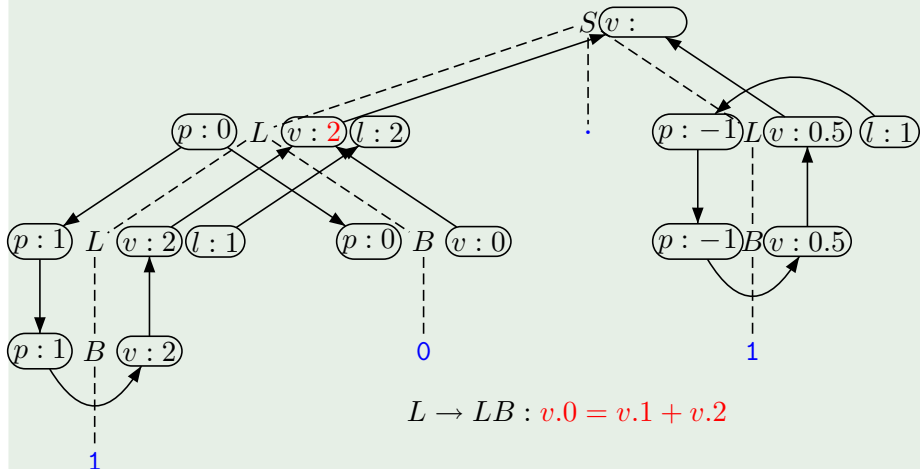
Attributed syntax tree for 10.1:



Adding Inherited Attributes II

Example 15.2 (continued)

Attributed syntax tree for 10.1:



- 1 Regular vs. $LR(0)$ Languages
- 2 Overview
- 3 Semantic Analysis
- 4 Attribute Grammars
- 5 Adding Inherited Attributes
- 6 Formal Definition of Attribute Grammars

Definition 15.3 (Attribute grammar)

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

Definition 15.3 (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 15.3 (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 15.3 (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 15.3 (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:

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

- 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 $f : V^{\alpha_1} \times \dots \times V^{\alpha_n} \rightarrow V^\alpha$.

Definition 15.3 (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:

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

- 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 $f : V^{\alpha_1} \times \dots \times V^{\alpha_n} \rightarrow V^\alpha$.

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

Definition 15.3 (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 15.4 (cf. Example 15.2)

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

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

Example 15.4 (cf. Example 15.2)

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

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

Example 15.4 (cf. Example 15.2)

$\mathcal{A}_B \in AG$ 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:**

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

Example 15.4 (cf. Example 15.2)

$\mathfrak{A}_B \in AG$ 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:**

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

- **Attribute variables:**

$\pi \in P$	$S \rightarrow L$	$S \rightarrow L.L$	$L \rightarrow B$
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	$\{v.1, v.2, l.1, p.0\}$	$\{p.0\}$	$\{p.0\}$

Example 15.4 (cf. Example 15.2)

$\mathcal{A}_B \in AG$ 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:**

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

- **Attribute variables:**

$\pi \in P$	$S \rightarrow L$	$S \rightarrow L.L$	$L \rightarrow B$
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 15.2
(e.g., $E_{S \rightarrow L} = \{v.0 = v.1, p.1 = 0\}$)