Compiler Construction

Lecture 12: Semantic Analysis I (Attribute Grammars)

Summer Semester 2016

Thomas Noll
Software Modeling and Verification Group
RWTH Aachen University

https://moves.rwth-aachen.de/teaching/ss-16/cc/

Overview

Outline of Lecture 12

Overview

Semantic Analysis

Attribute Grammars

Adding Inherited Attributes

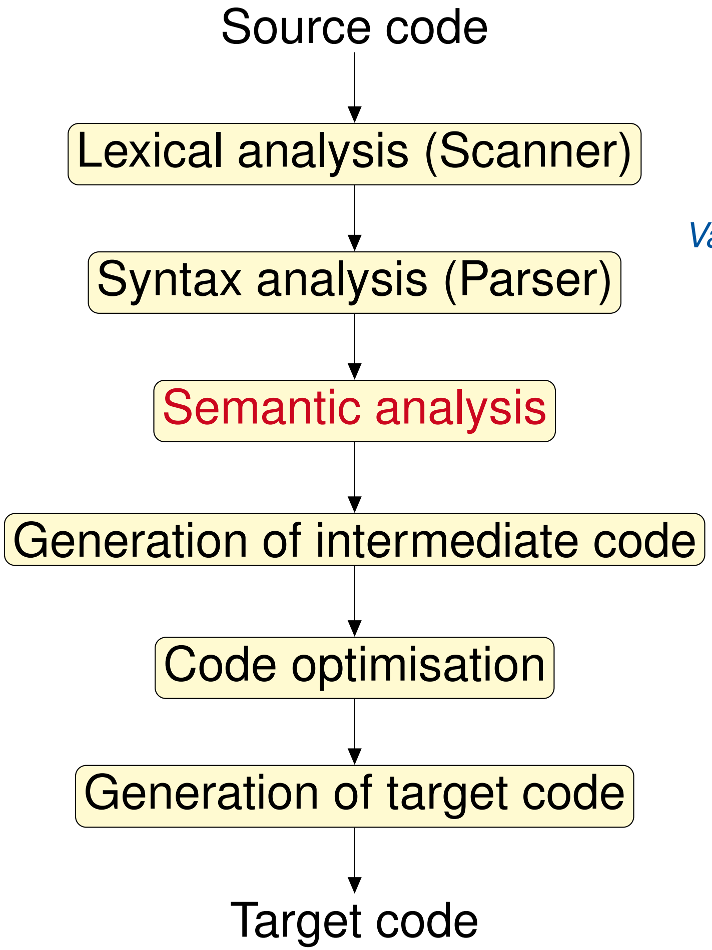
Formal Definition of Attribute Grammars

The Attribute Equation System

Circularity of Attribute Grammars

Overview

Conceptual Structure of a Compiler



Source code

Lexical analysis (Scanner)

Syntax analysis (Parser)

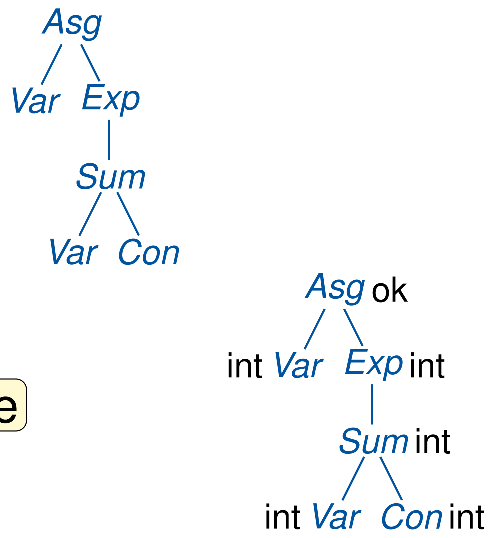
Semantic analysis

Generation of intermediate code

Code optimisation

Generation of target code

Target code



attribute grammars

Outline of Lecture 12

Overview

Semantic Analysis

Attribute Grammars

Adding Inherited Attributes

Formal Definition of Attribute Grammars

The Attribute Equation System

Circularity of Attribute Grammars

Beyond Syntax

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

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

Semantic Analysis

Beyond Syntax

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?

Semantic Analysis

Beyond Syntax

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?

Semantic Analysis

Beyond Syntax

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?

Semantic Analysis

Beyond Syntax

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?

Beyond Syntax

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)?

Semantic Analysis

Beyond Syntax

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)?
- ...

Semantic Analysis

Beyond Syntax

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!

Semantic Analysis

Beyond Syntax

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!

(For example, $\{ww \mid w \in \Sigma^*\} \notin CFL_\Sigma$)

Semantic Analysis

Static 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).

Static 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, ...

Attribute Grammars

Outline of Lecture 12

Overview

Semantic Analysis

Attribute Grammars

Adding Inherited Attributes

Formal Definition of Attribute Grammars

The Attribute Equation System

Circularity 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

$\Longrightarrow$ 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

$\Longrightarrow$ 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.

Attribute Grammars II

Advantage: attribute grammars provide a very flexible and broadly applicable mechanism for transporting information throught 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 12.1 (only synthesized attributes)

Binary numbers (with fraction):

$$\begin{array}{lll} G_B: & \text{Numbers} & S \to L \\ & & S \to L\,.\,L \\ & \text{Lists} & L \to B \\ & & \\ & & L \to LB \\ & & \\ & \text{Bits} & B \to 0 \\ & \text{Bits} & B \to 1 \end{array}$$

Example: Knuth’s Binary Numbers I

Example 12.1 (only synthesized attributes)

Binary numbers (with fraction):

$$
\begin{array}{llllcl}
G_B: & \text{Numbers} & S \to L & d.0 & = & d.1 \\
 & & S \to L\,.\,L & d.0 & = & d.1 + d.3/2^{l.3} \\
 & \text{Lists} & L \to B & d.0 & = & d.1 \\
 & & & l.0 & = & 1 \\
 & & L \to LB & d.0 & = & 2 * d.1 + d.2 \\
 & & & l.0 & = & l.1 + 1 \\
 & \text{Bits} & B \to 0 & d.0 & = & 0 \\
 & \text{Bits} & B \to 1 & d.0 & = & 1
\end{array}
$$

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

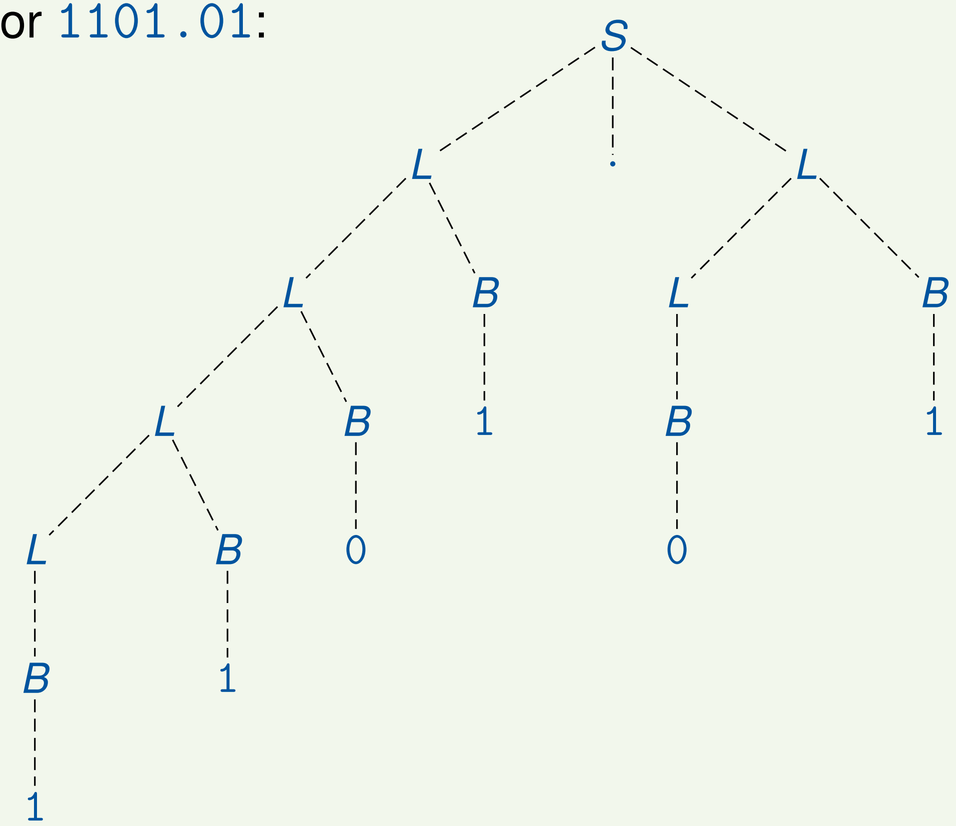
Semantic rules: equations with attribute variables (index = position; 0 = LHS)

Attribute Grammars

Example: Knuth's Binary Numbers II

Example 12.1 (continued)

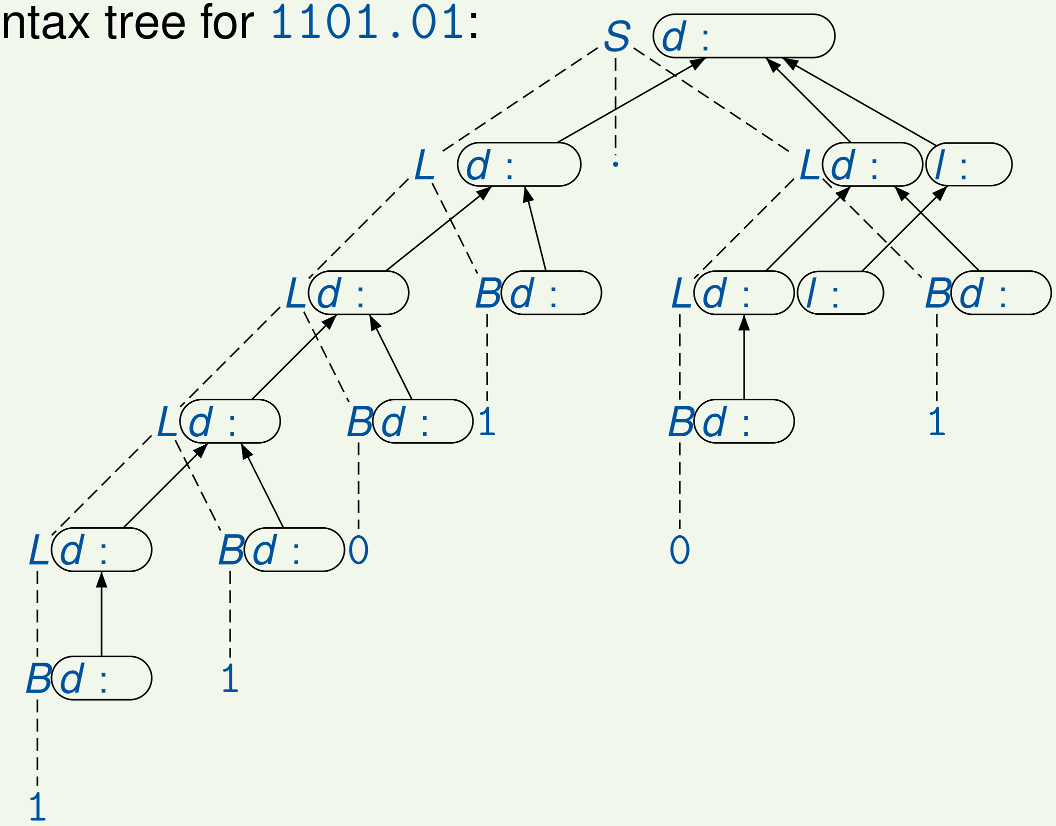
Syntax tree for `1101.01`:



Example: Knuth’s Binary Numbers II

Example 12.1 (continued)

Attributed syntax tree for `1101.01`:

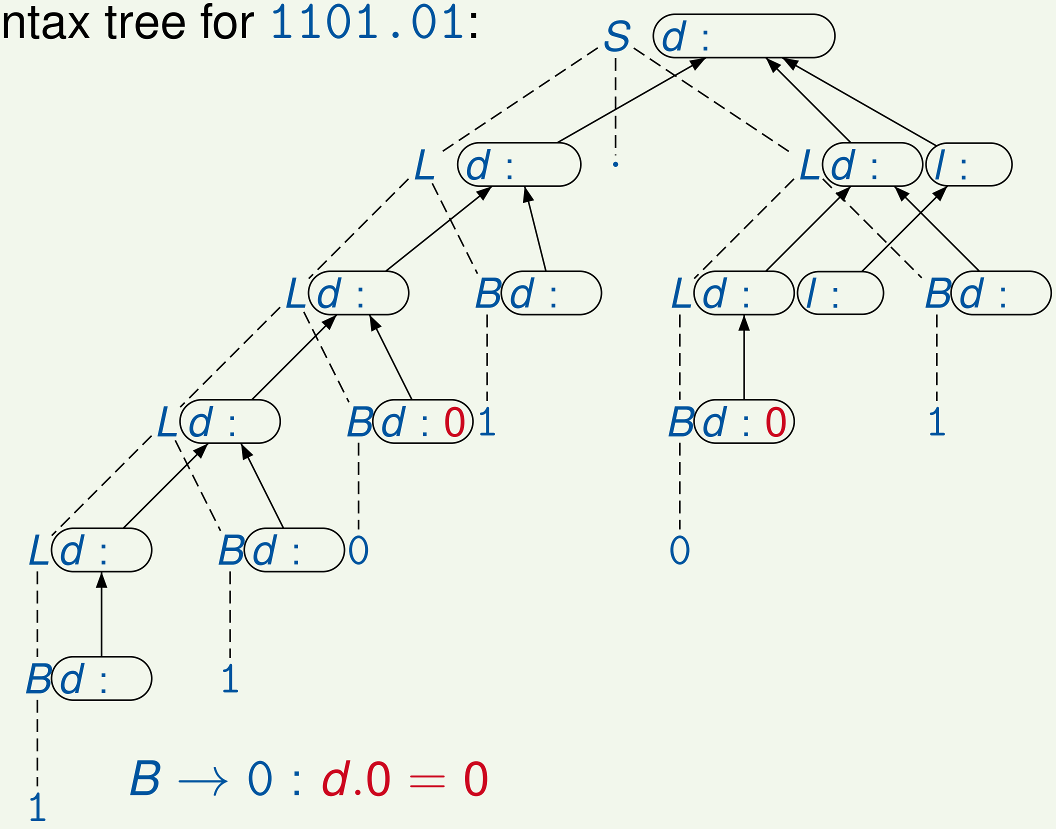


Attribute Grammars

Example: Knuth's Binary Numbers II

Example 12.1 (continued)

Attributed syntax tree for `1101.01`:

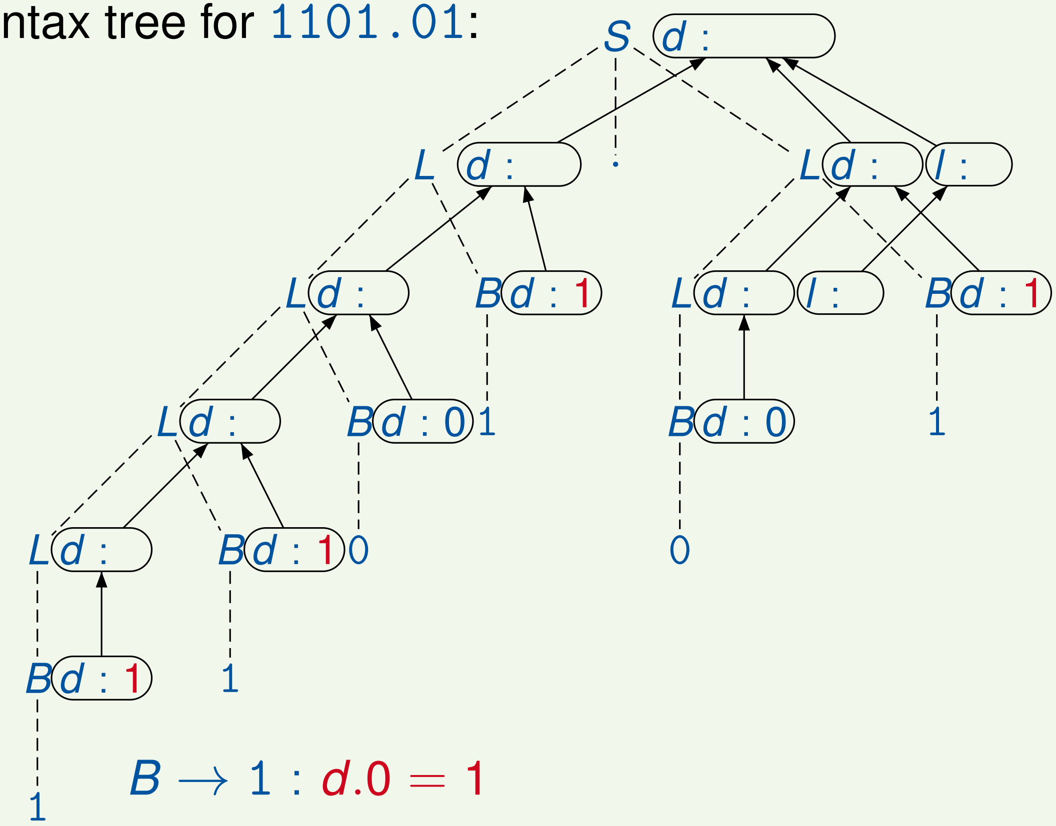


$$B \rightarrow 0 : d.0 = 0$$

Example: Knuth's Binary Numbers II

Example 12.1 (continued)

Attributed syntax tree for `1101.01`:



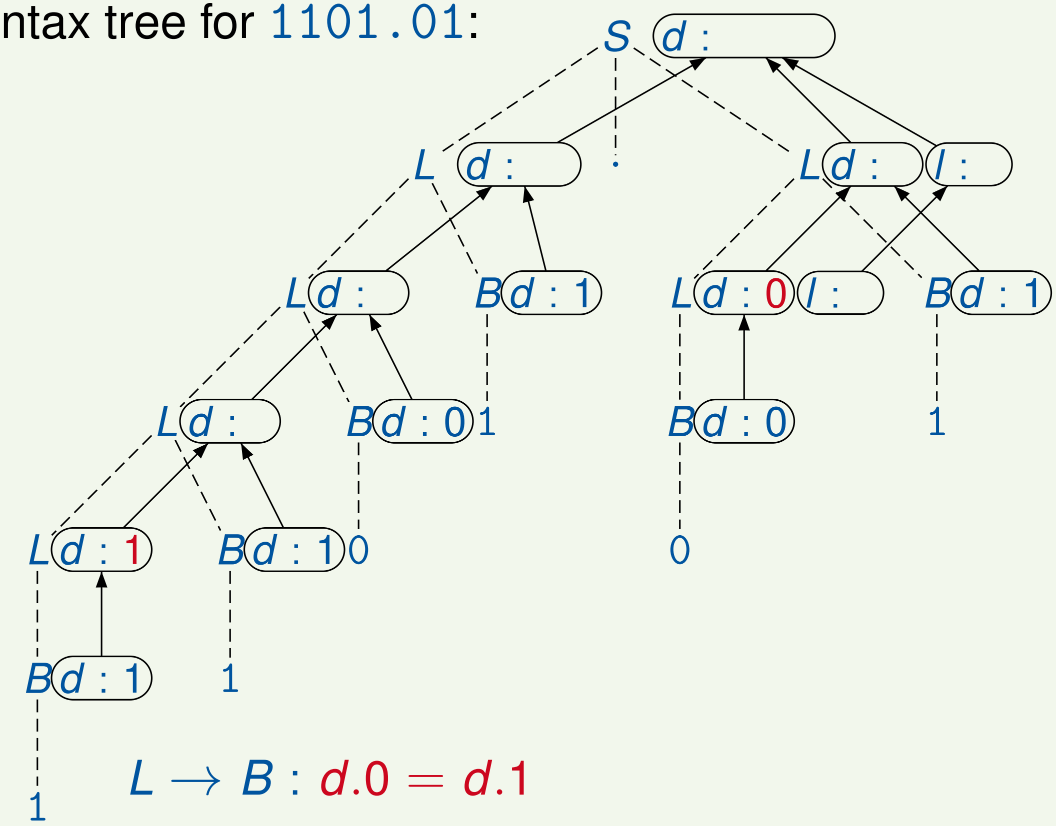
$$B \to 1 : d.0 = 1$$

Attribute Grammars

Example: Knuth’s Binary Numbers II

Example 12.1 (continued)

Attributed syntax tree for `1101.01`:

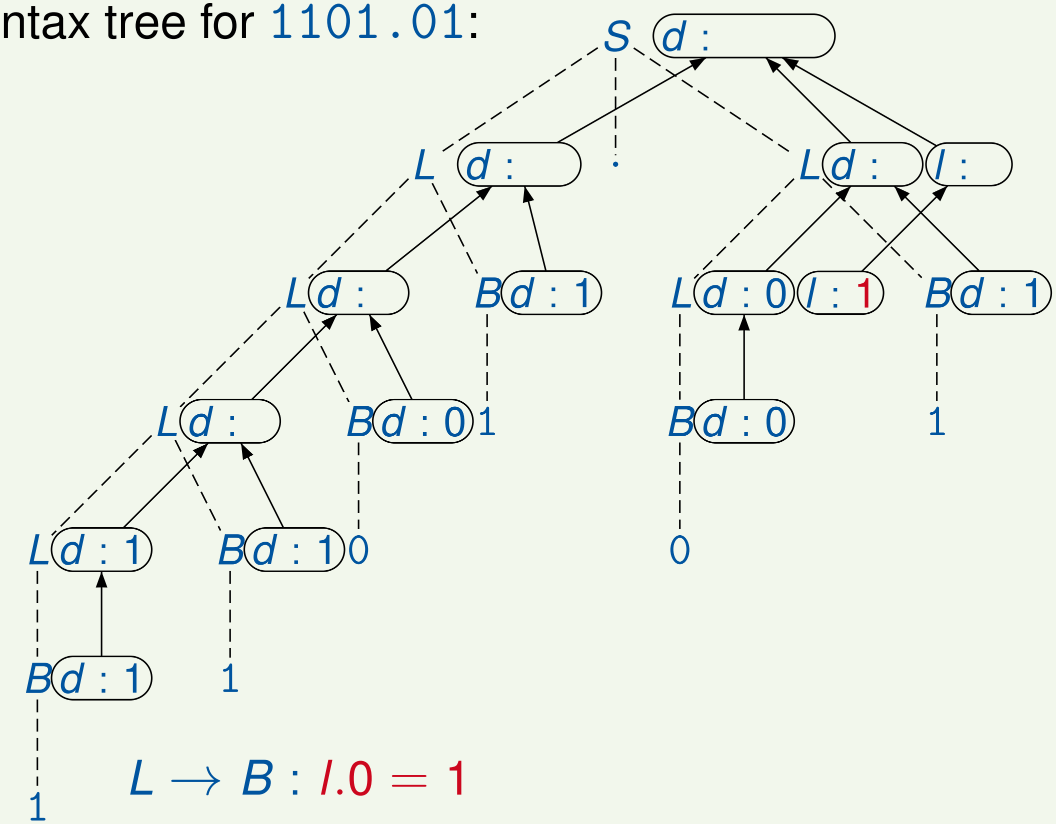


$$L \rightarrow B : d.0 = d.1$$

Example: Knuth's Binary Numbers II

Example 12.1 (continued)

Attributed syntax tree for `1101.01`:



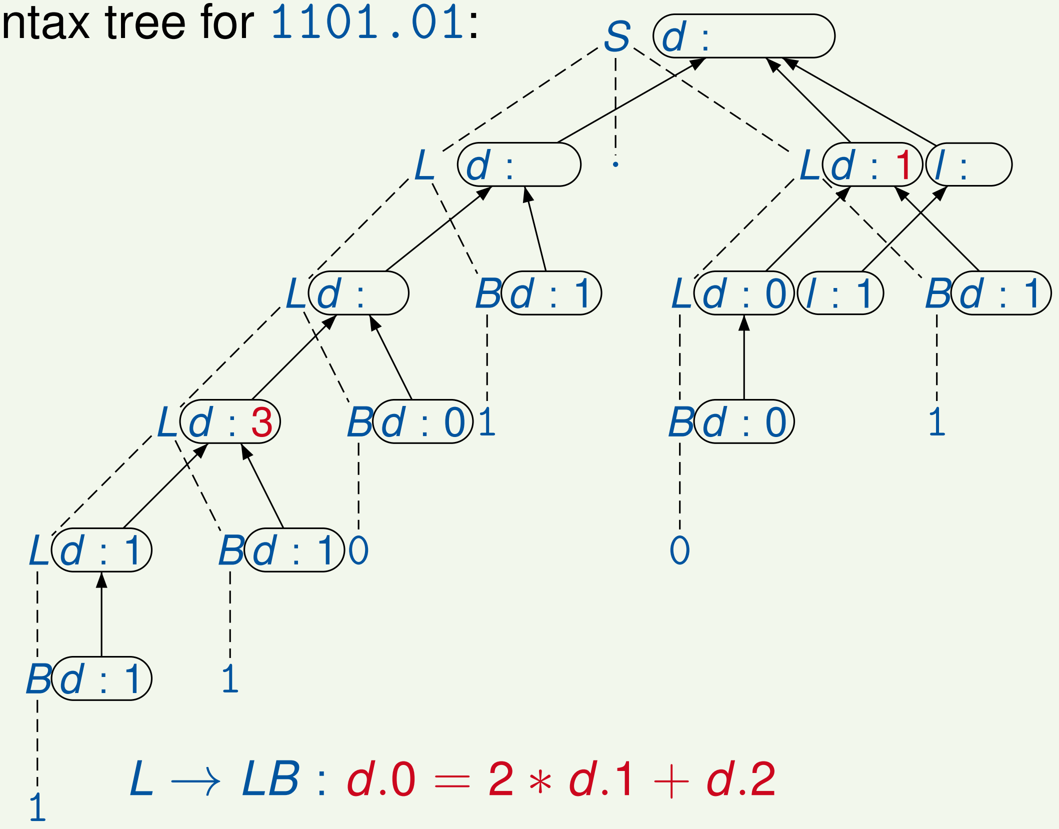
$L \rightarrow B : l.0 = 1$

Attribute Grammars

Example: Knuth’s Binary Numbers II

Example 12.1 (continued)

Attributed syntax tree for `1101.01`:

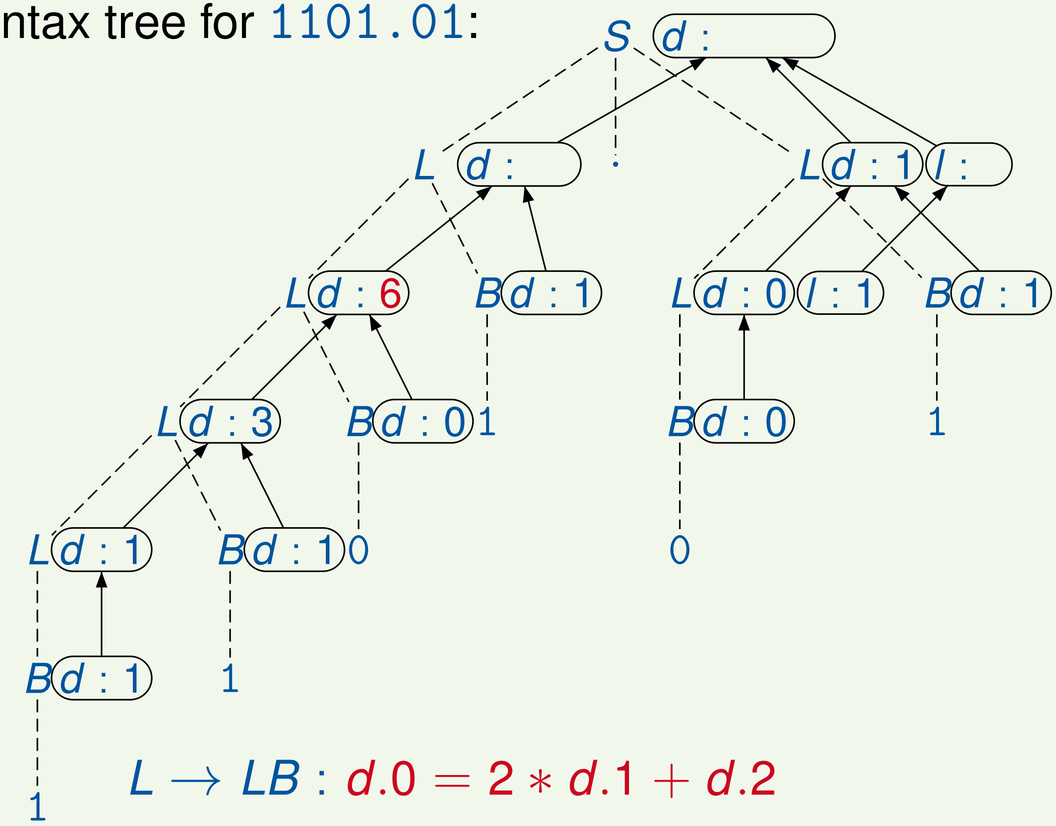


$$L \rightarrow LB : d.0 = 2 * d.1 + d.2$$

Example: Knuth's Binary Numbers II

Example 12.1 (continued)

Attributed syntax tree for `1101.01`:

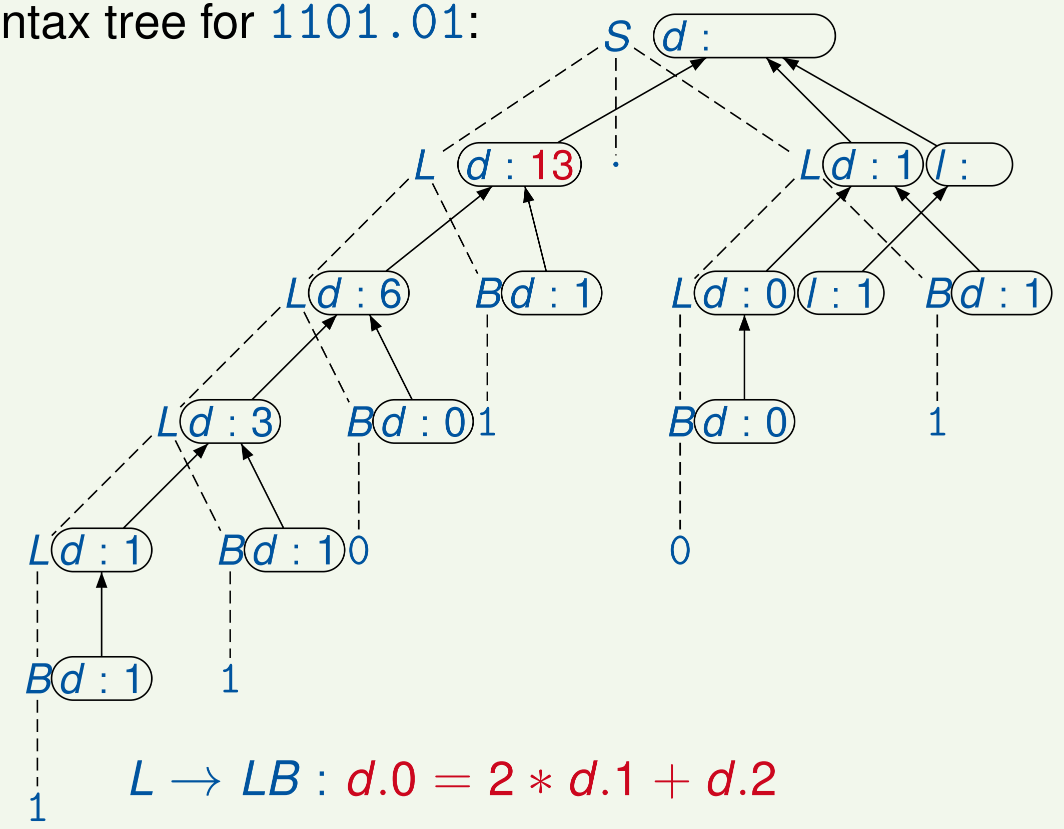


$$L \rightarrow LB : d.0 = 2 * d.1 + d.2$$

Example: Knuth’s Binary Numbers II

Example 12.1 (continued)

Attributed syntax tree for `1101.01`:



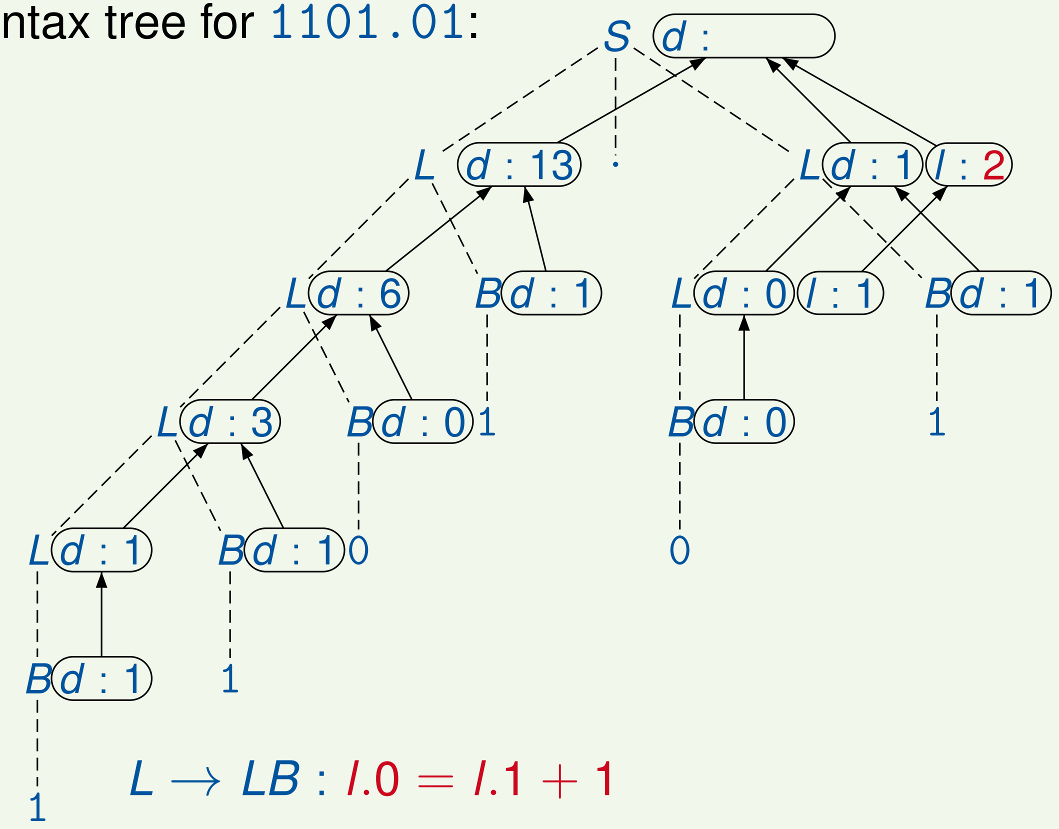
$$L \to LB : d.0 = 2 * d.1 + d.2$$

Attribute Grammars

Example: Knuth’s Binary Numbers II

Example 12.1 (continued)

Attributed syntax tree for `1101.01`:

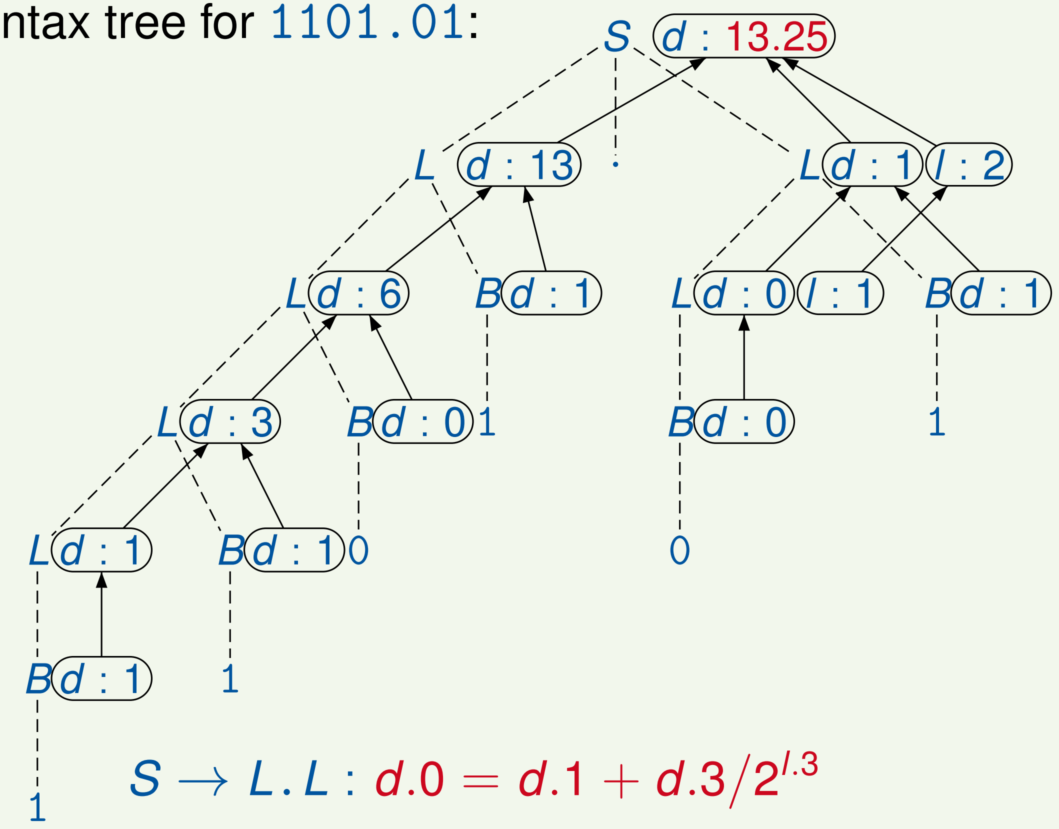


$$L \rightarrow LB : l.0 = l.1 + 1$$

Example: Knuth's Binary Numbers II

Example 12.1 (continued)

Attributed syntax tree for `1101.01`:



$$S \to L\,.\,L : d.0 = d.1 + d.3/2^{l.3}$$

Adding Inherited Attributes

Outline of Lecture 12

Adding Inherited Attributes I

Example 12.2 (synthesized + inherited attributes)

Binary numbers (with fraction):

G'_B :	Numbers	$S \to L$ $S \to L\,.\,L$
	Lists	$L \to B$
		$L \to LB$
	Bits	$B \to 0$
	Bits	$B \to 1$

Adding Inherited Attributes I

Example 12.2 (synthesized + inherited attributes)

Binary numbers (with fraction):

$$
\begin{array}{lllrclrcl}
G'_B: & \text{Numbers} & S \to L & d.0 & = & d.1 & p.1 & = & 0 \\
 & & S \to L\,.\,L & d.0 & = & d.1 + d.3 & & & \\
 & & & p.1 & = & 0 & p.3 & = & -l.3 \\
 & \text{Lists} & L \to B & d.0 & = & d.1 & l.0 & = & 1 \\
 & & & p.1 & = & p.0 & & & \\
 & & L \to LB & d.0 & = & d.1 + d.2 & l.0 & = & l.1 + 1 \\
 & & & p.1 & = & p.0 + 1 & p.2 & = & p.0 \\
 & \text{Bits} & B \to 0 & d.0 & = & 0 & & & \\
 & \text{Bits} & B \to 1 & d.0 & = & 2^{p.0} & & &
\end{array}
$$

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

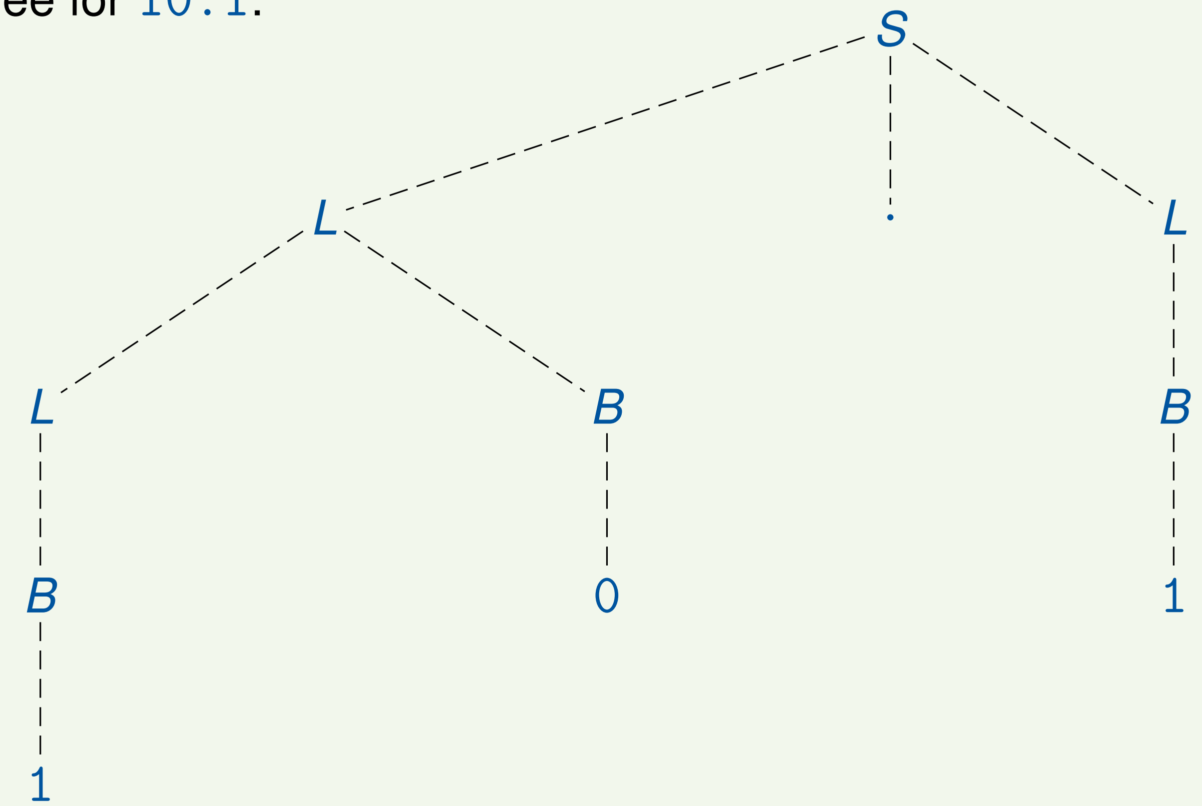
Inherited attribute of L, B: p (position; domain: $V^p := \mathbb{Z}$)

Adding Inherited Attributes

Adding Inherited Attributes II

Example 12.2 (continued)

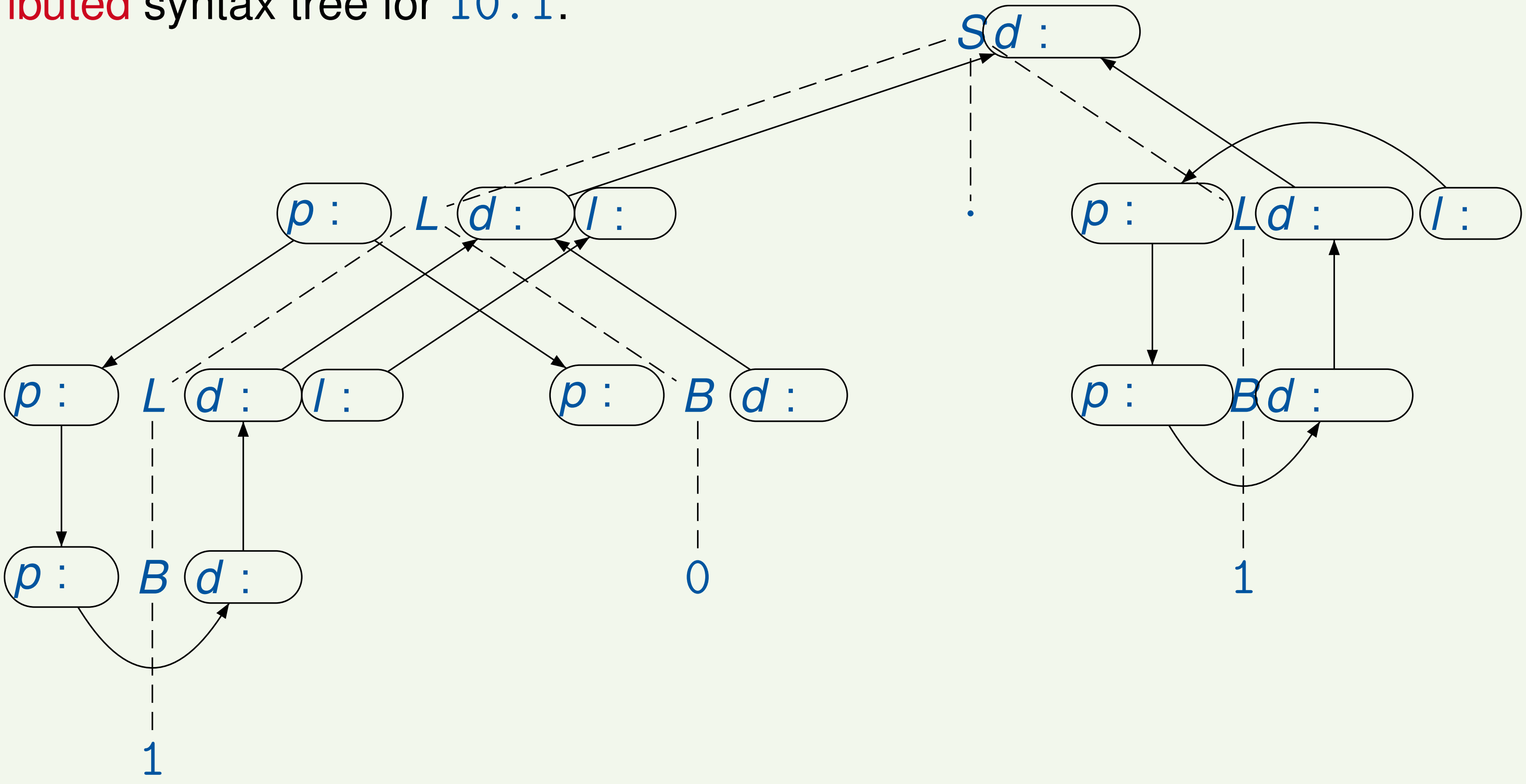
Syntax tree for `10.1`:



Adding Inherited Attributes II

Example 12.2 (continued)

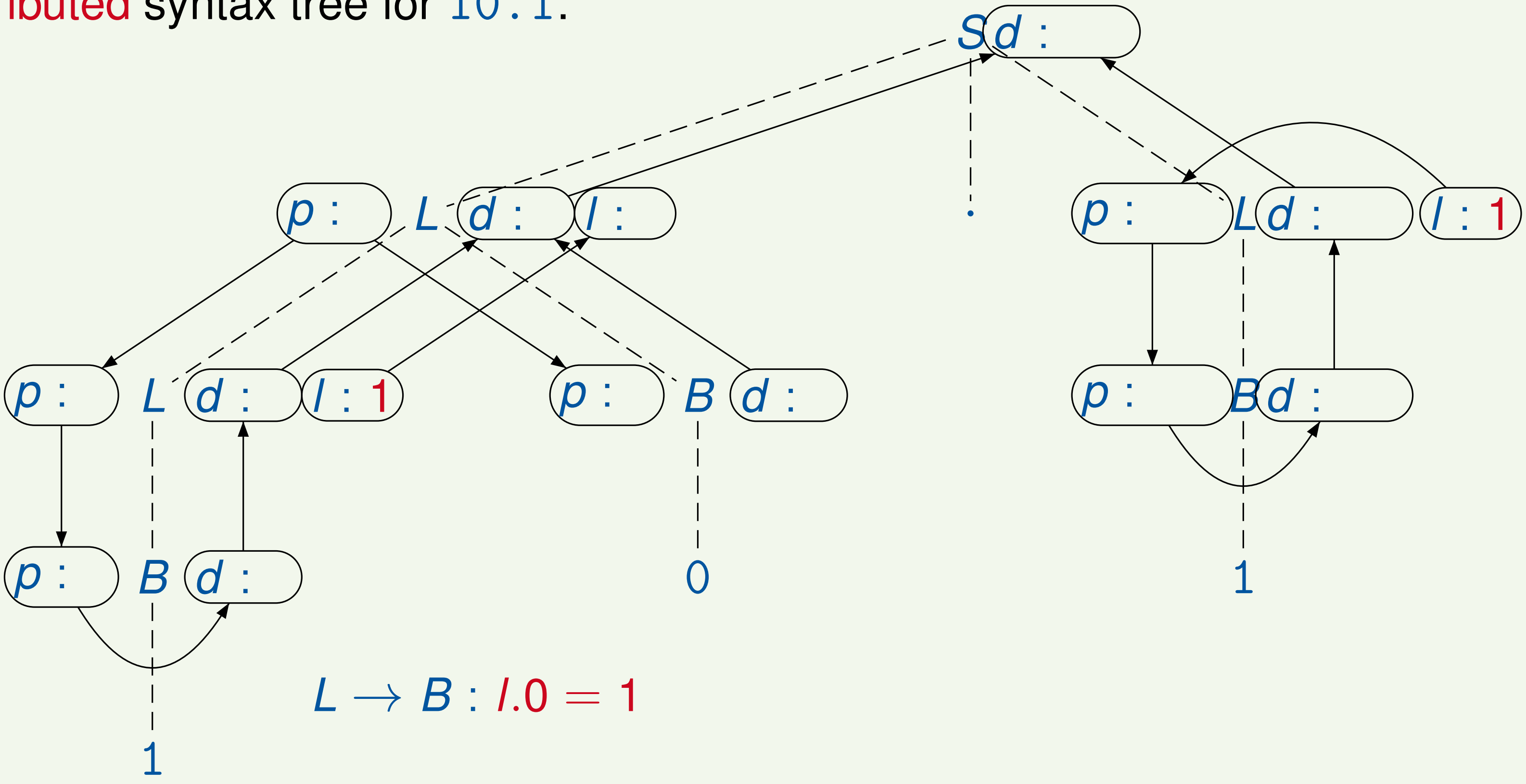
Attributed syntax tree for `10.1`:



Adding Inherited Attributes II

Example 12.2 (continued)

Attributed syntax tree for `10.1`:

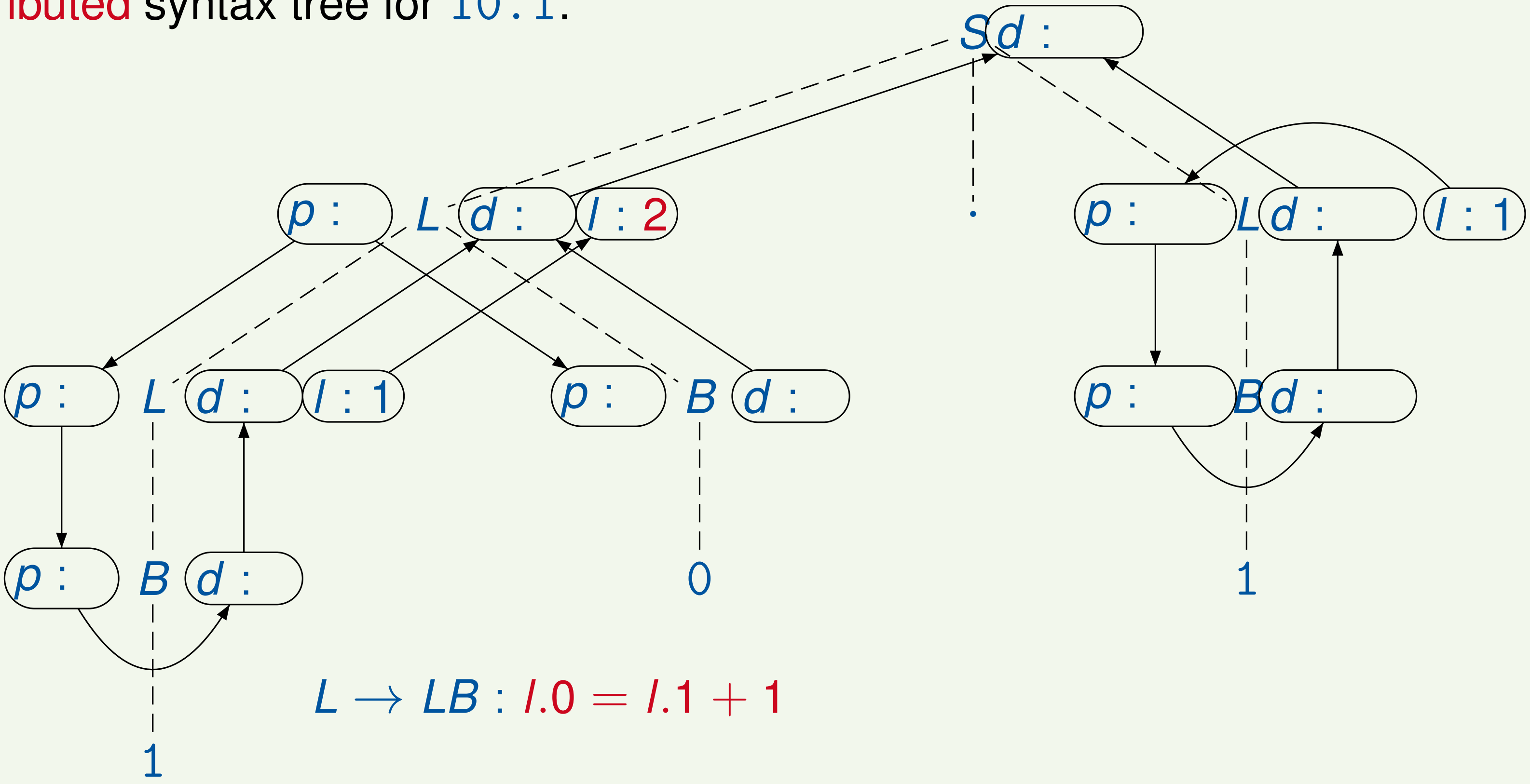


$L \rightarrow B : l.0 = 1$

Adding Inherited Attributes II

Example 12.2 (continued)

Attributed syntax tree for `10.1`:

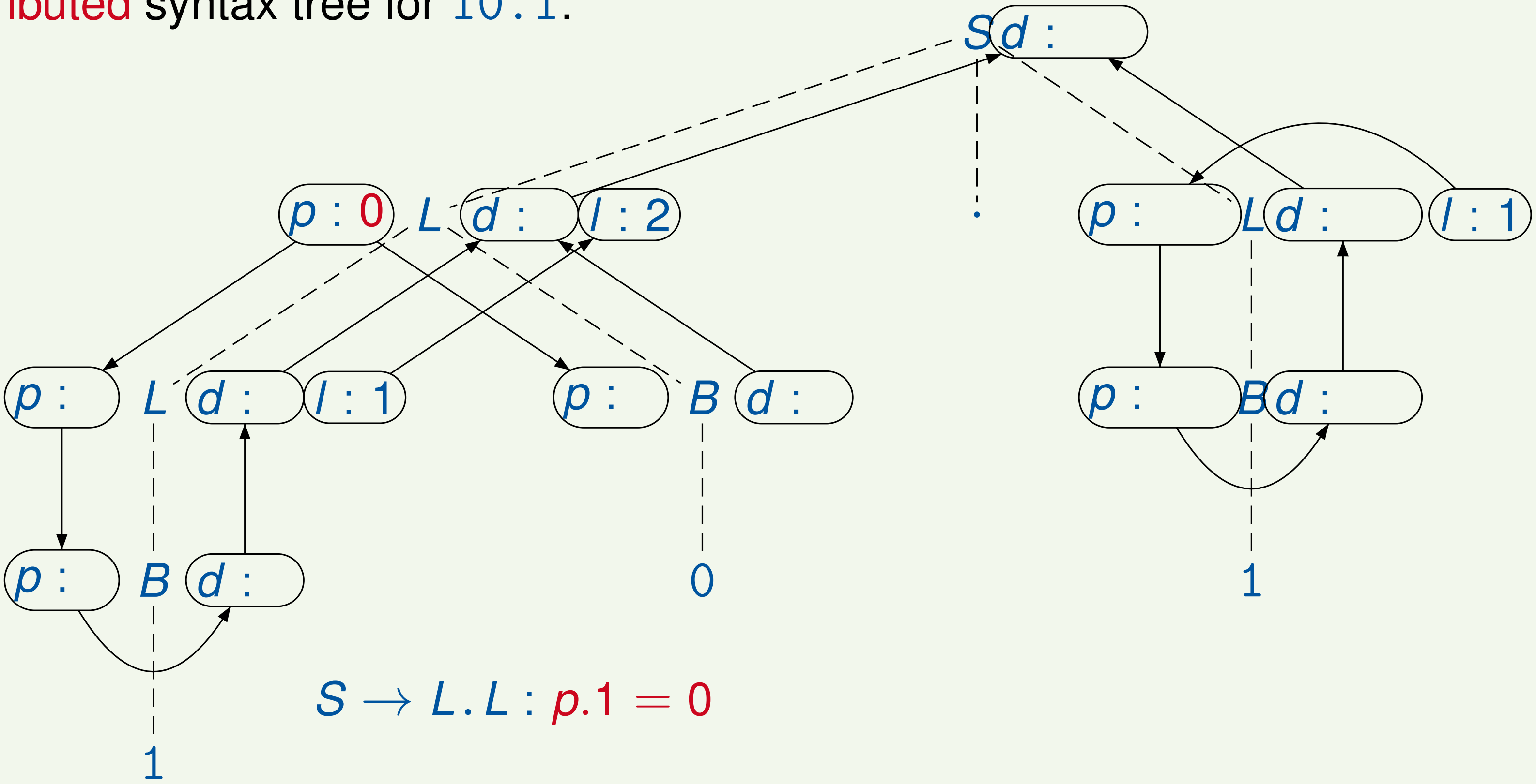


$L \rightarrow LB : l.0 = l.1 + 1$

Adding Inherited Attributes II

Example 12.2 (continued)

Attributed syntax tree for `10.1`:



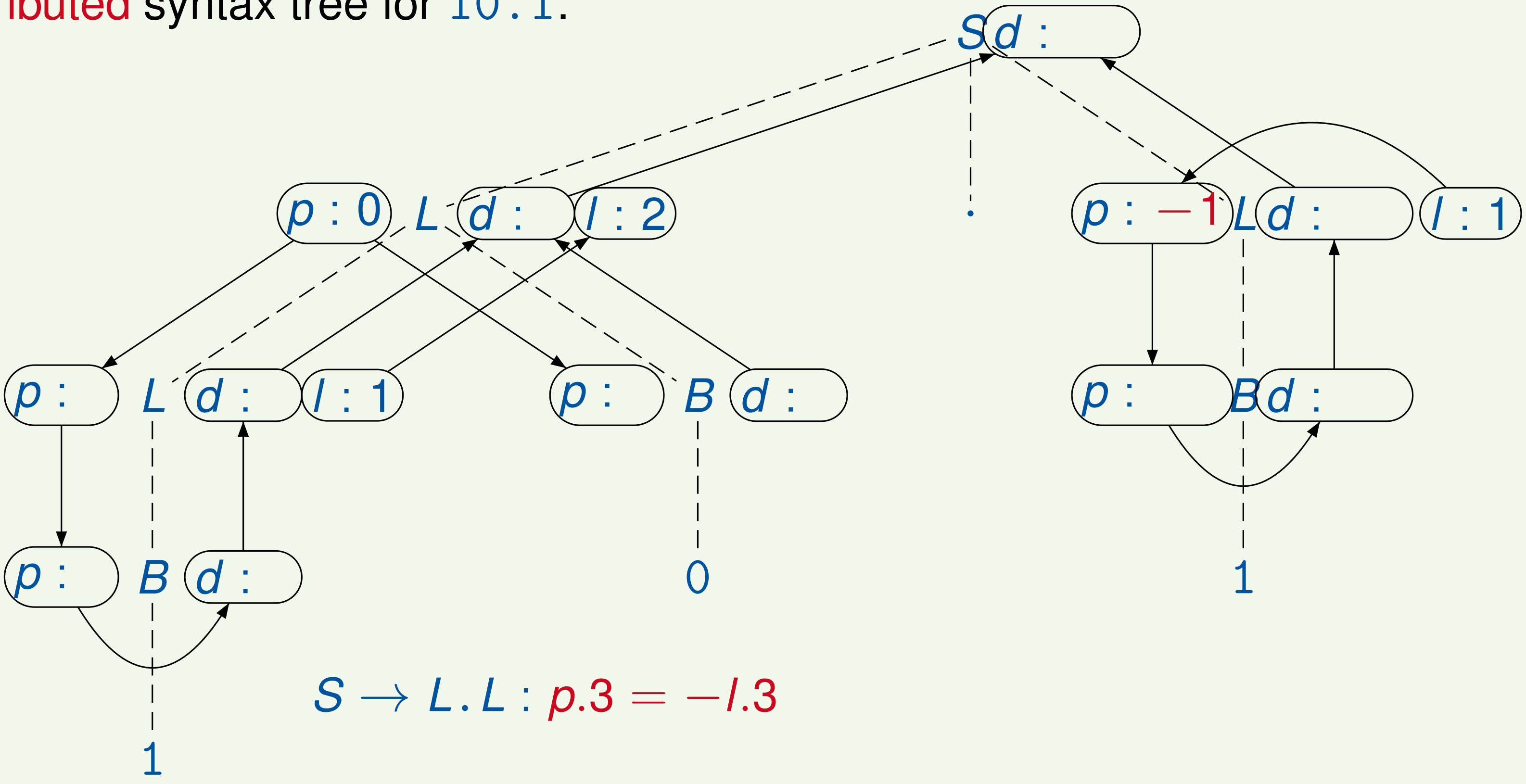
$S \rightarrow L\,.\,L : p.1 = 0$

Adding Inherited Attributes

Adding Inherited Attributes II

Example 12.2 (continued)

Attributed syntax tree for `10.1`:

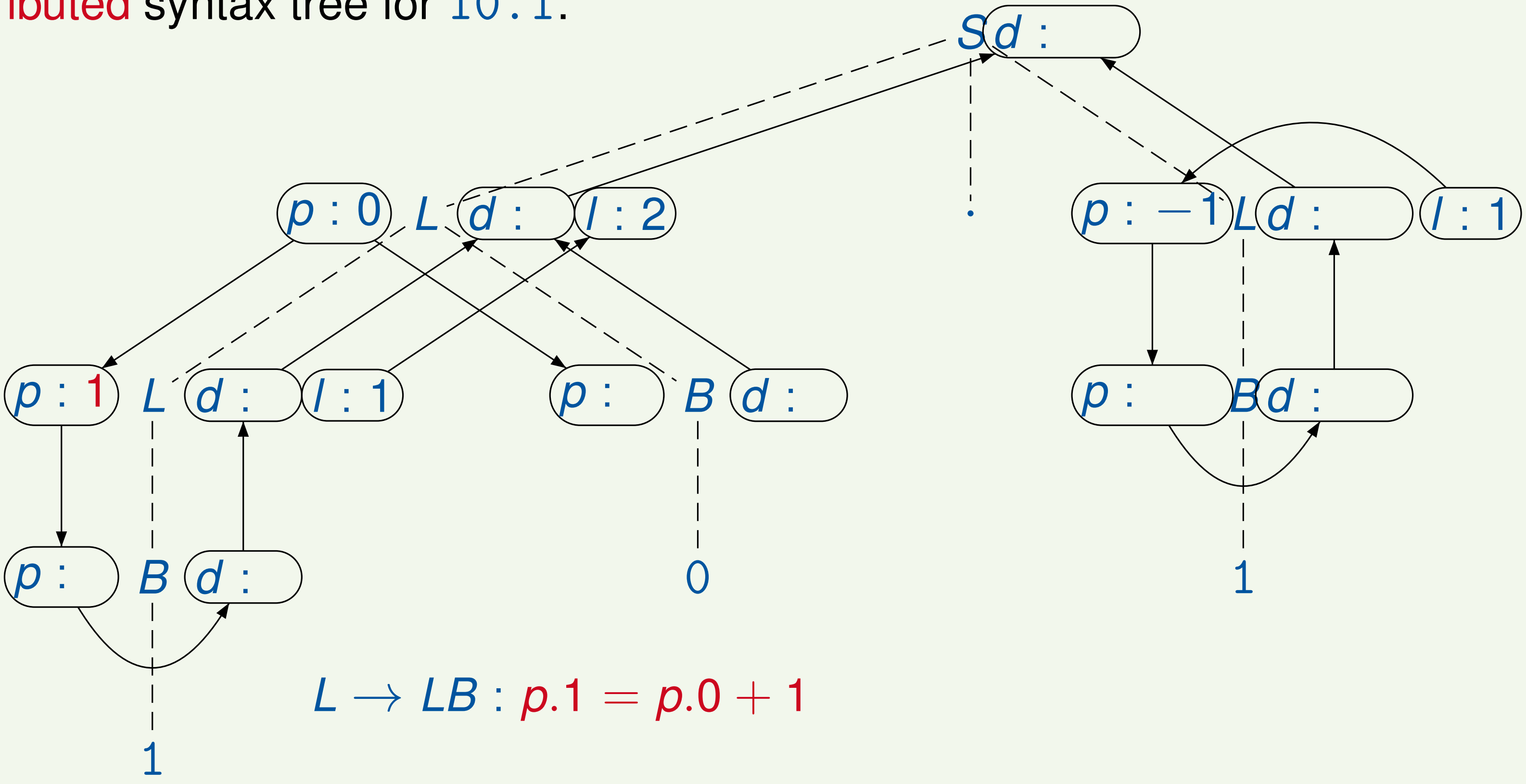


$S \rightarrow L\,.\,L : p.3 = -l.3$

Adding Inherited Attributes II

Example 12.2 (continued)

Attributed syntax tree for `10.1`:



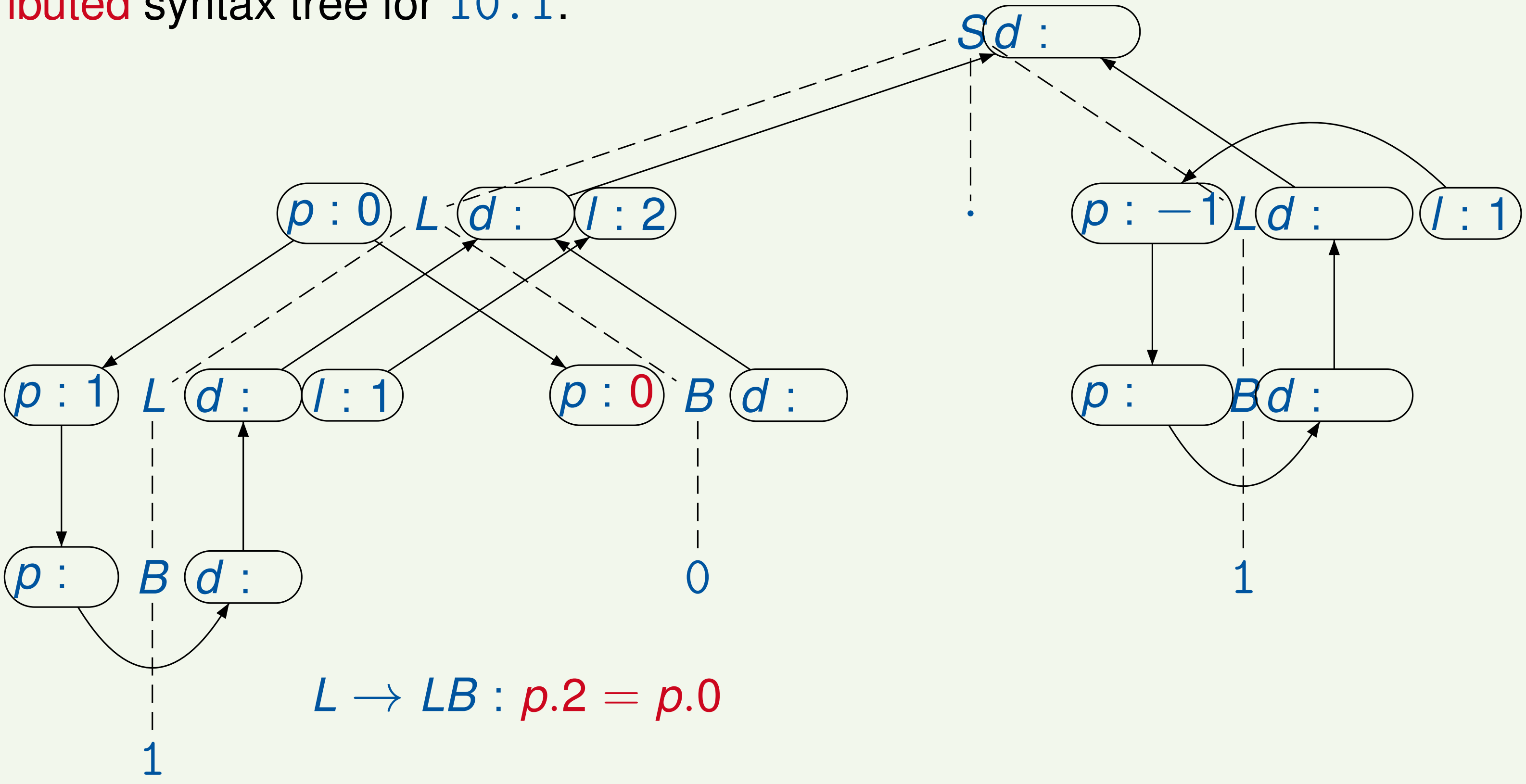
$L \rightarrow LB : p.1 = p.0 + 1$

Adding Inherited Attributes

Adding Inherited Attributes II

Example 12.2 (continued)

Attributed syntax tree for `10.1`:

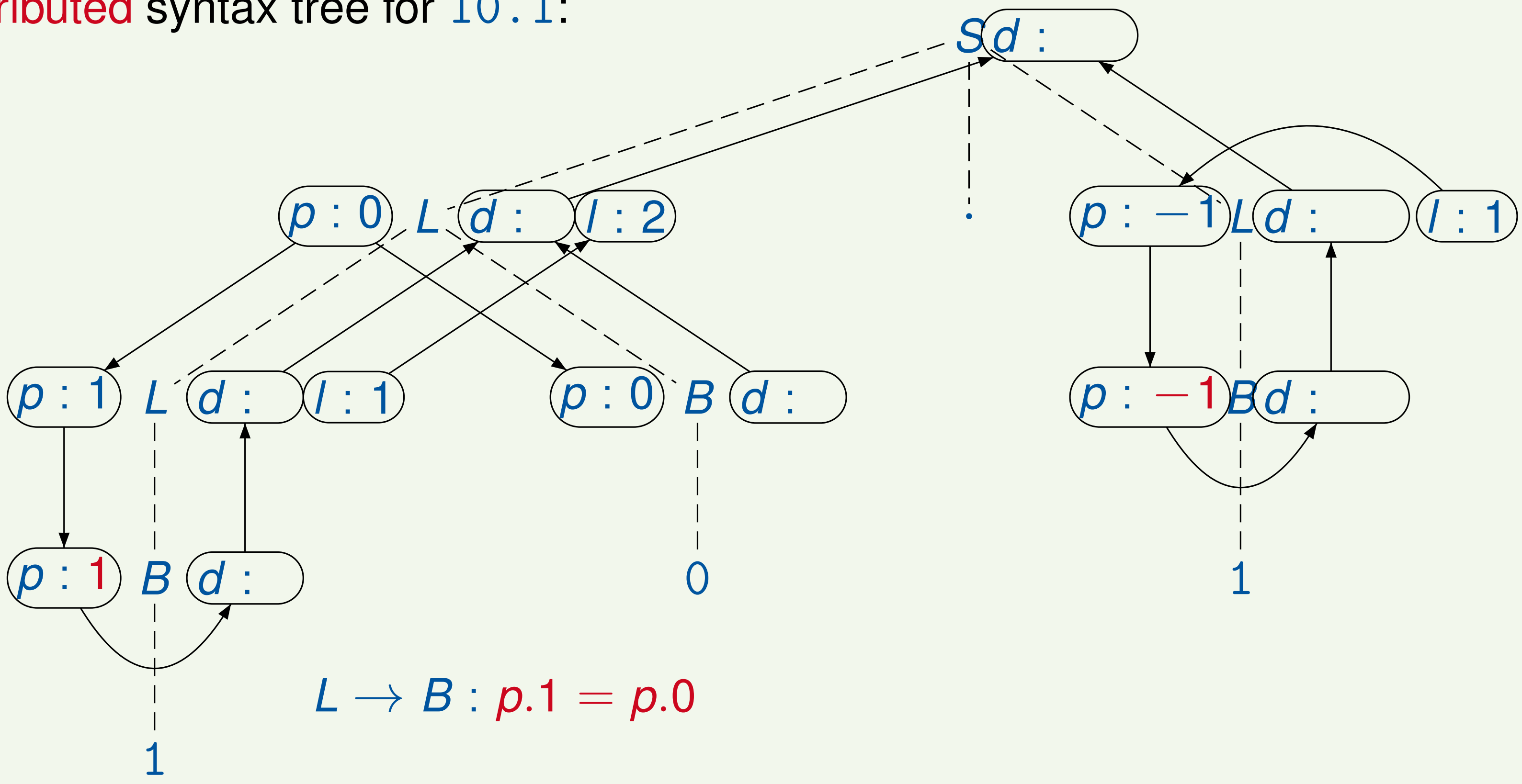


$L \to LB : p.2 = p.0$

Adding Inherited Attributes II

Example 12.2 (continued)

Attributed syntax tree for `10.1`:

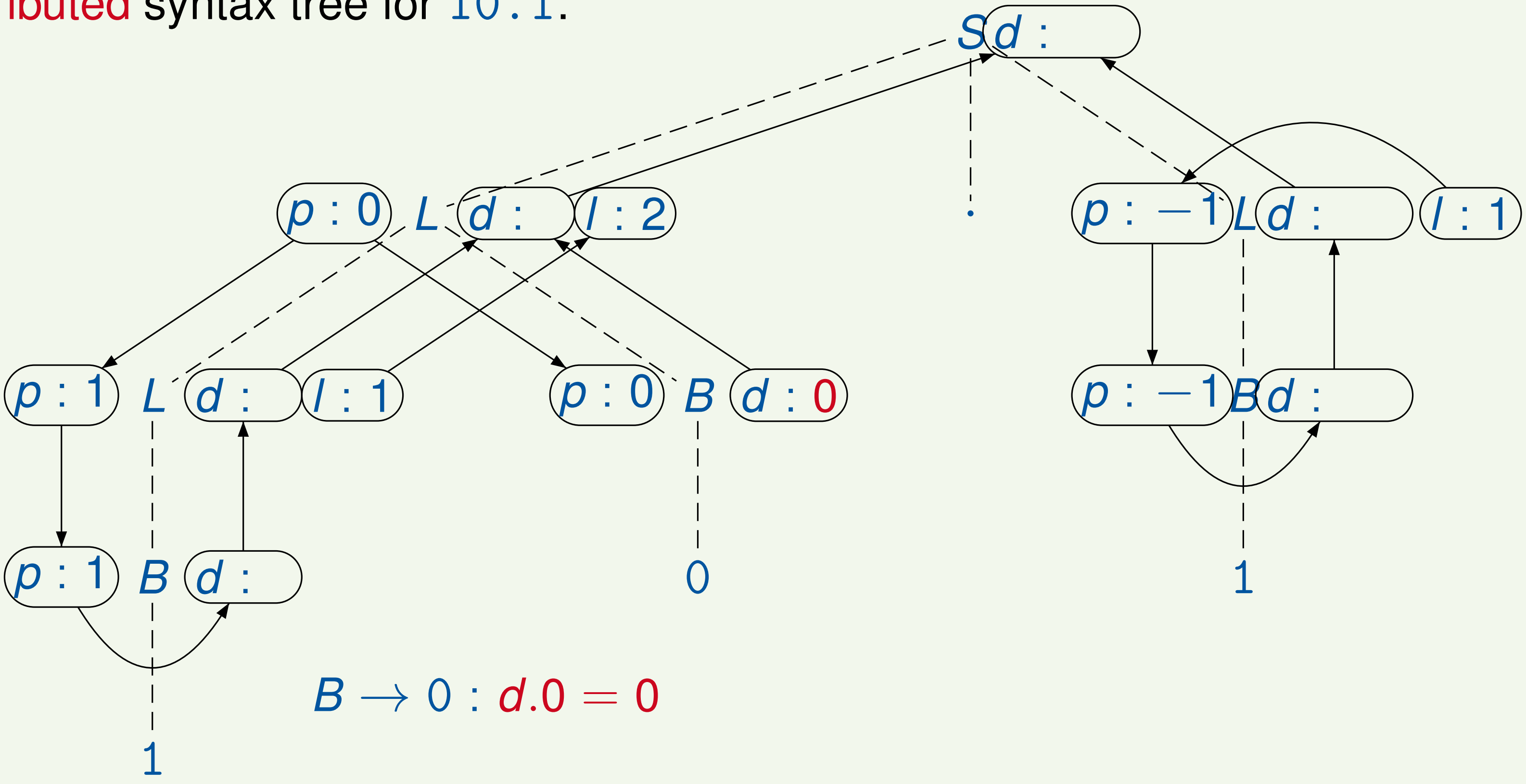


$L \to B : p.1 = p.0$

Adding Inherited Attributes II

Example 12.2 (continued)

Attributed syntax tree for `10.1`:

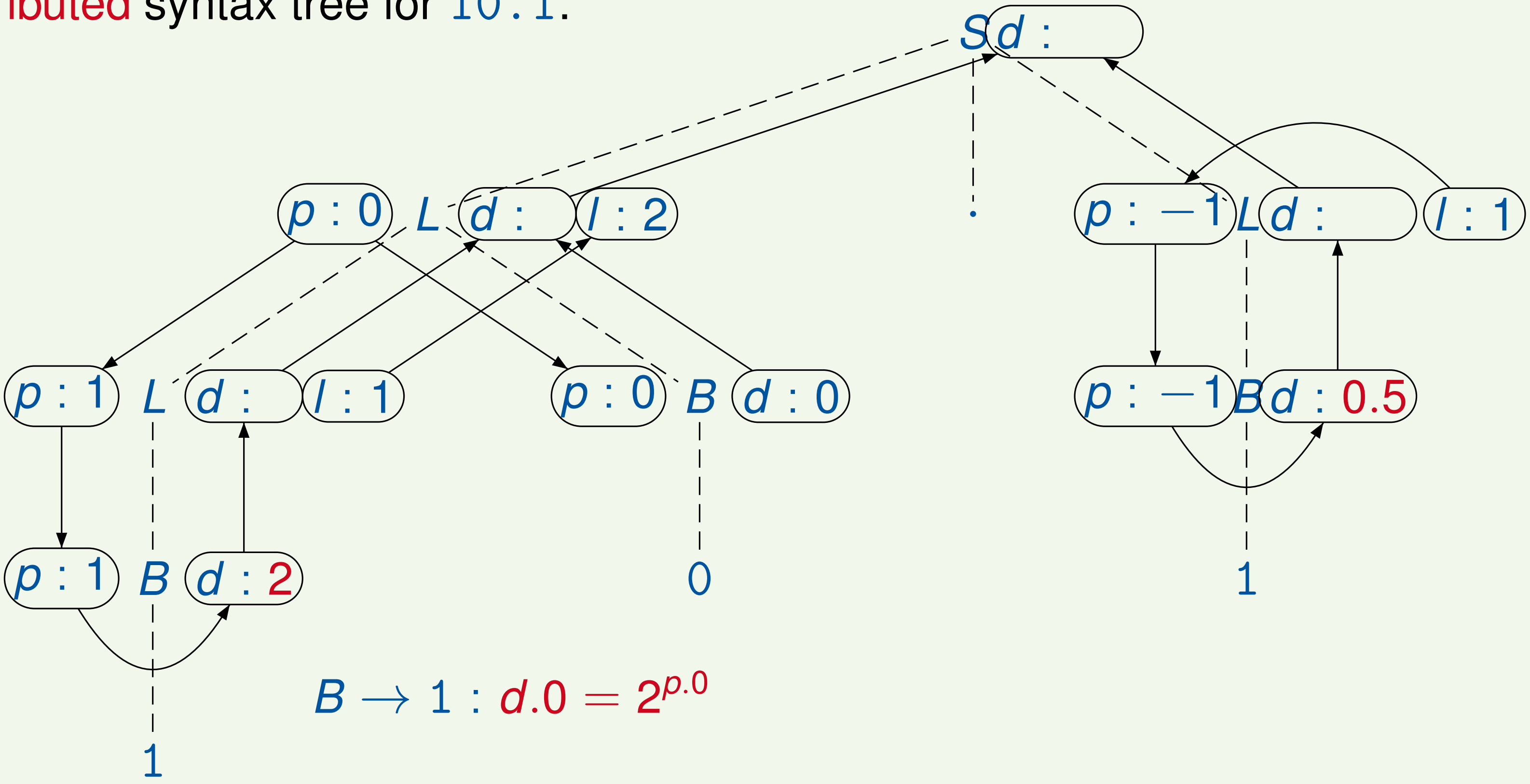


$B \to 0 : d.0 = 0$

Adding Inherited Attributes II

Example 12.2 (continued)

Attributed syntax tree for `10.1`:



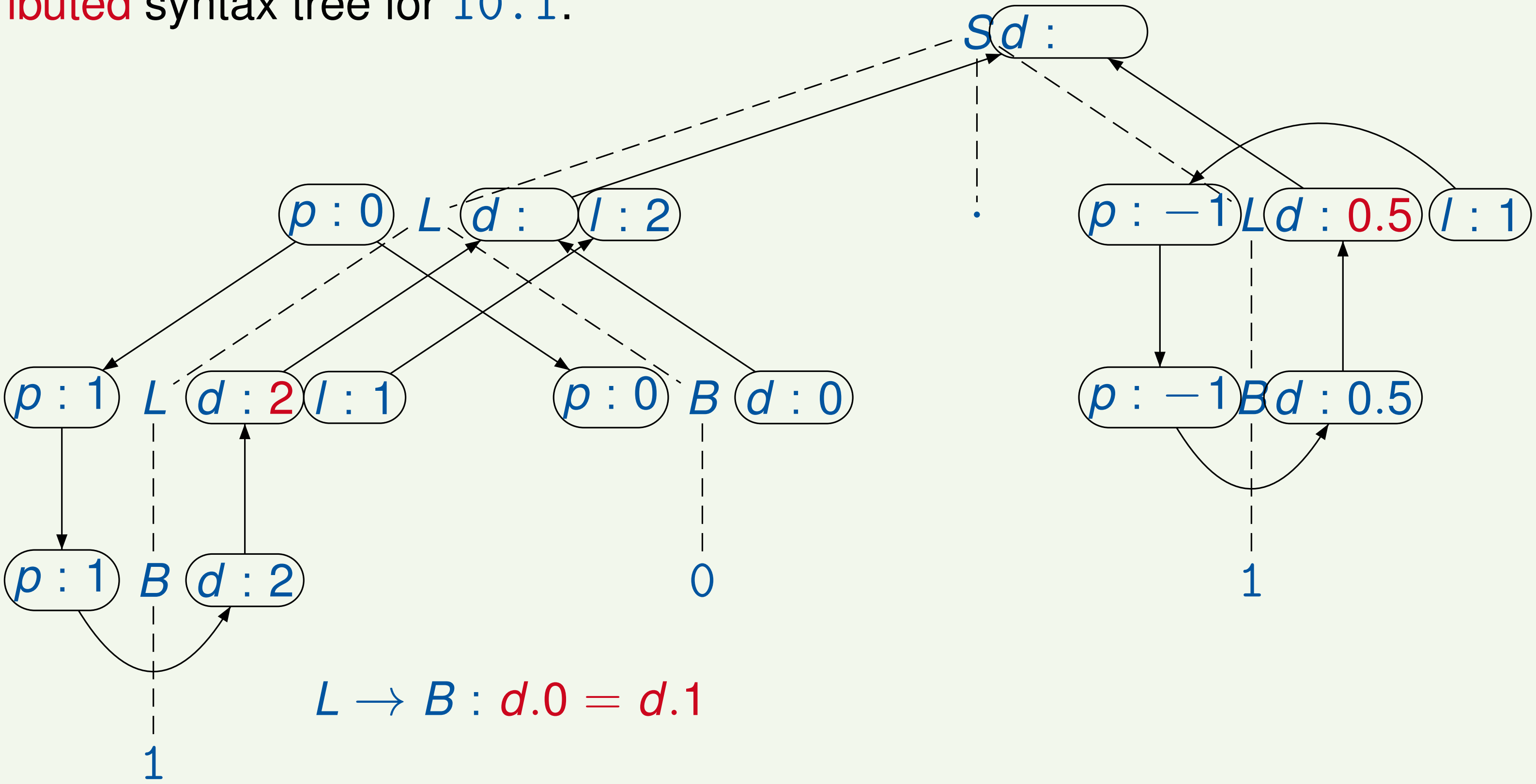
$B \rightarrow 1 : d.0 = 2^{p.0}$

Adding Inherited Attributes

Adding Inherited Attributes II

Example 12.2 (continued)

Attributed syntax tree for `10.1`:



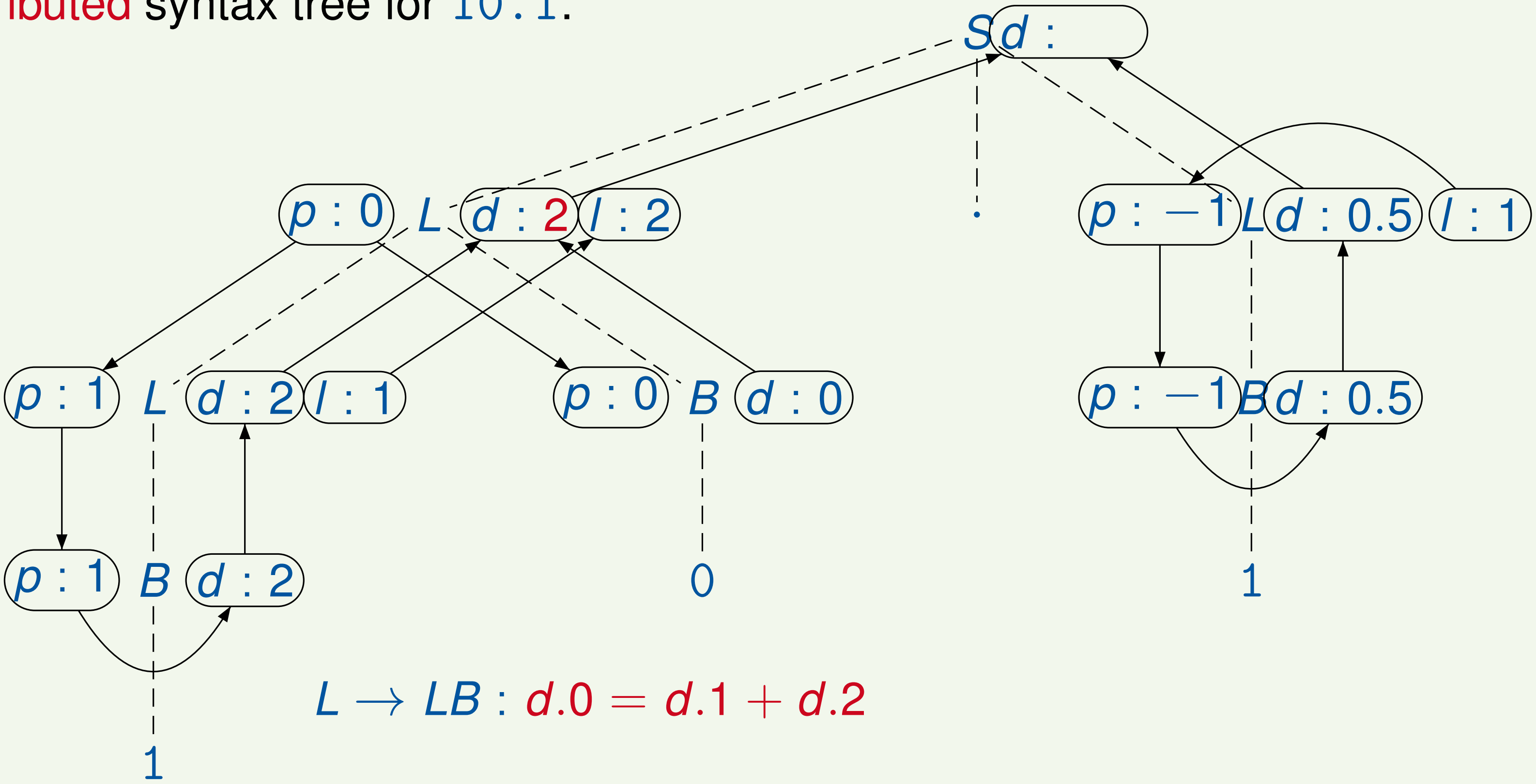
$L \rightarrow B : d.0 = d.1$

Adding Inherited Attributes

Adding Inherited Attributes II

Example 12.2 (continued)

Attributed syntax tree for `10.1`:

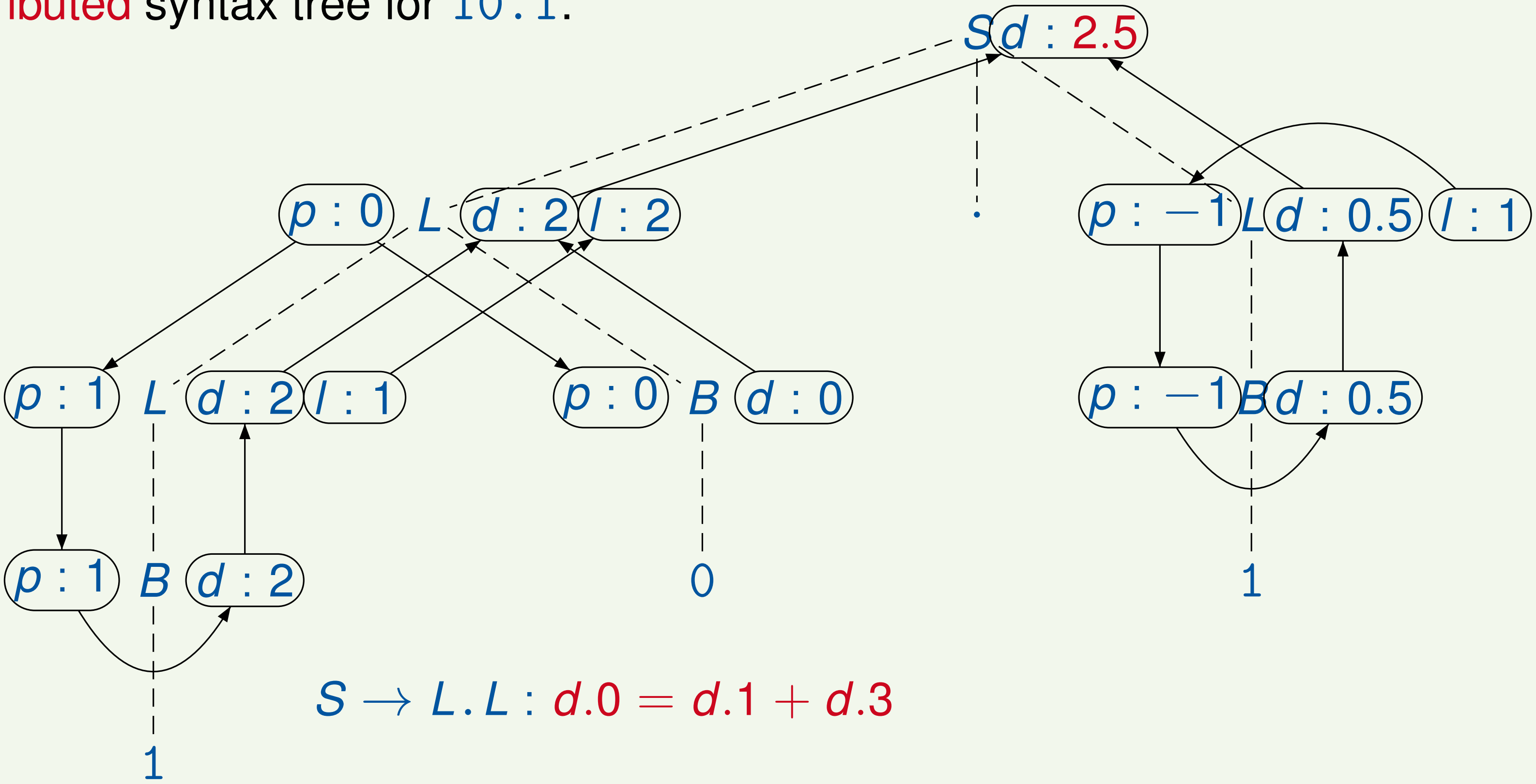


$L \rightarrow LB : d.0 = d.1 + d.2$

Adding Inherited Attributes II

Example 12.2 (continued)

Attributed syntax tree for `10.1`:



$S \to L.L : d.0 = d.1 + d.3$

Formal Definition of Attribute Grammars

Outline of Lecture 12

Overview

Semantic Analysis

Attribute Grammars

Adding Inherited Attributes

Formal Definition of Attribute Grammars

The Attribute Equation System

Circularity of Attribute Grammars

Formal Definition of Attribute Grammars

Formal Definition of Attribute Grammars I

Definition 12.3 (Attribute grammar)

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

Formal Definition of Attribute Grammars

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.

Formal Definition of Attribute Grammars

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 $\mathrm{att} : X \to 2^{Att}$ be an attribute assignment, and let $\mathrm{syn}(Y) := \mathrm{att}(Y) \cap Syn$ and $\mathrm{inh}(Y) := \mathrm{att}(Y) \cap Inh$ for every $Y \in X$.

Formal Definition of Attribute Grammars

Formal Definition of Attribute Grammars I

Definition 12.3 (Attribute grammar)

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

- Let $\mathit{Att} = \mathit{Syn} \uplus \mathit{Inh}$ be a set of (synthesized or inherited) attributes, and let $V = \bigcup_{\alpha \in \mathit{Att}} V^\alpha$ be a union of value sets.
- Let $\mathrm{att} : X \to 2^{\mathit{Att}}$ be an attribute assignment, and let $\mathrm{syn}(Y) := \mathrm{att}(Y) \cap \mathit{Syn}$ and $\mathrm{inh}(Y) := \mathrm{att}(Y) \cap \mathit{Inh}$ for every $Y \in X$.
- Every production $\pi = Y_0 \to Y_1 \ldots Y_r \in P$ determines the set
  $$\mathit{Var}_\pi := \{\alpha.i \mid \alpha \in \mathrm{att}(Y_i), i \in \{0, \ldots, r\}\}$$
  of attribute variables of π with the subsets of inner and outer variables:
  $$\mathit{In}_\pi := \{\alpha.i \mid (i = 0, \alpha \in \mathrm{syn}(Y_i)) \text{ or } (i \in [r], \alpha \in \mathrm{inh}(Y_i))\}, \qquad \mathit{Out}_\pi := \mathit{Var}_\pi \setminus \mathit{In}_\pi$$

Formal Definition of Attribute Grammars

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

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 $\mathrm{att} : X \to 2^{Att}$ be an attribute assignment, and let $\mathrm{syn}(Y) := \mathrm{att}(Y) \cap Syn$ and $\mathrm{inh}(Y) := \mathrm{att}(Y) \cap Inh$ for every $Y \in X$.
- Every production $\pi = Y_0 \to Y_1 \dots Y_r \in P$ determines the set
  $$Var_\pi := \{\alpha.i \mid \alpha \in \mathrm{att}(Y_i), i \in \{0, \dots, r\}\}$$
  of attribute variables of π with the subsets of inner and outer variables:
  $$In_\pi := \{\alpha.i \mid (i = 0, \alpha \in \mathrm{syn}(Y_i)) \text{ or } (i \in [r], \alpha \in \mathrm{inh}(Y_i))\}, \qquad Out_\pi := Var_\pi \setminus In_\pi$$
- 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 $f : V^{\alpha_1} \times \dots \times V^{\alpha_n} \to V^\alpha$.
- For each $\pi \in P$, let E_π be a set with exactly one semantic rule for every inner variable of π, and let $E := (E_\pi \mid \pi \in P)$.

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

Then $\mathfrak{A} := \langle G, E, V \rangle$ is called an attribute grammar: $\mathfrak{A} \in AG$.

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 $Inh = \{p\}$

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: $V^d = \mathbb{Q}$, $V^l = \mathbb{N}$, $V^p = \mathbb{Z}$

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	.
$\mathrm{syn}(Y)$	$\{d\}$	$\{d, l\}$	$\{d\}$	$\emptyset$	$\emptyset$	$\emptyset$
$\mathrm{inh}(Y)$	$\emptyset$	$\{p\}$	$\{p\}$	$\emptyset$	$\emptyset$	$\emptyset$

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	$.$
$\mathrm{syn}(Y)$	$\{d\}$	$\{d, l\}$	$\{d\}$	$\emptyset$	$\emptyset$	$\emptyset$
$\mathrm{inh}(Y)$	$\emptyset$	$\{p\}$	$\{p\}$	$\emptyset$	$\emptyset$	$\emptyset$

- Attribute variables:

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

$\pi \in P$	$L \to LB$	$B \to 0$	$B \to 1$
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	$.$
$\mathrm{syn}(Y)$	$\{d\}$	$\{d, l\}$	$\{d\}$	$\emptyset$	$\emptyset$	$\emptyset$
$\mathrm{inh}(Y)$	$\emptyset$	$\{p\}$	$\{p\}$	$\emptyset$	$\emptyset$	$\emptyset$

- Attribute variables:

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

$\pi \in P$	$L \to LB$	$B \to 0$	$B \to 1$
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., $E_{S \to L} = \{d.0 = d.1, p.1 = 0\}$)

The Attribute Equation System

Outline of Lecture 12

Overview

Semantic Analysis

Attribute Grammars

Adding Inherited Attributes

Formal Definition of Attribute Grammars

The Attribute Equation System

Circularity of Attribute Grammars

Attribution of Syntax Trees I

Definition 12.5 (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:

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

The Attribute Equation System

Attribution of Syntax Trees I

Definition 12.5 (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:

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

- Let $k_0 \in K$ be an (inner) node where production $\pi = Y_0 \to Y_1 \dots Y_r \in P$ 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

Attribution of Syntax Trees I

Definition 12.5 (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:

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

- Let $k_0 \in K$ be an (inner) node where production $\pi = Y_0 \to Y_1 \ldots Y_r \in P$ is applied, and let $k_1, \ldots, k_r \in K$ 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 $i \in \{0, \ldots, r\}$ by k_i.
- The attribute equation system of t is given by

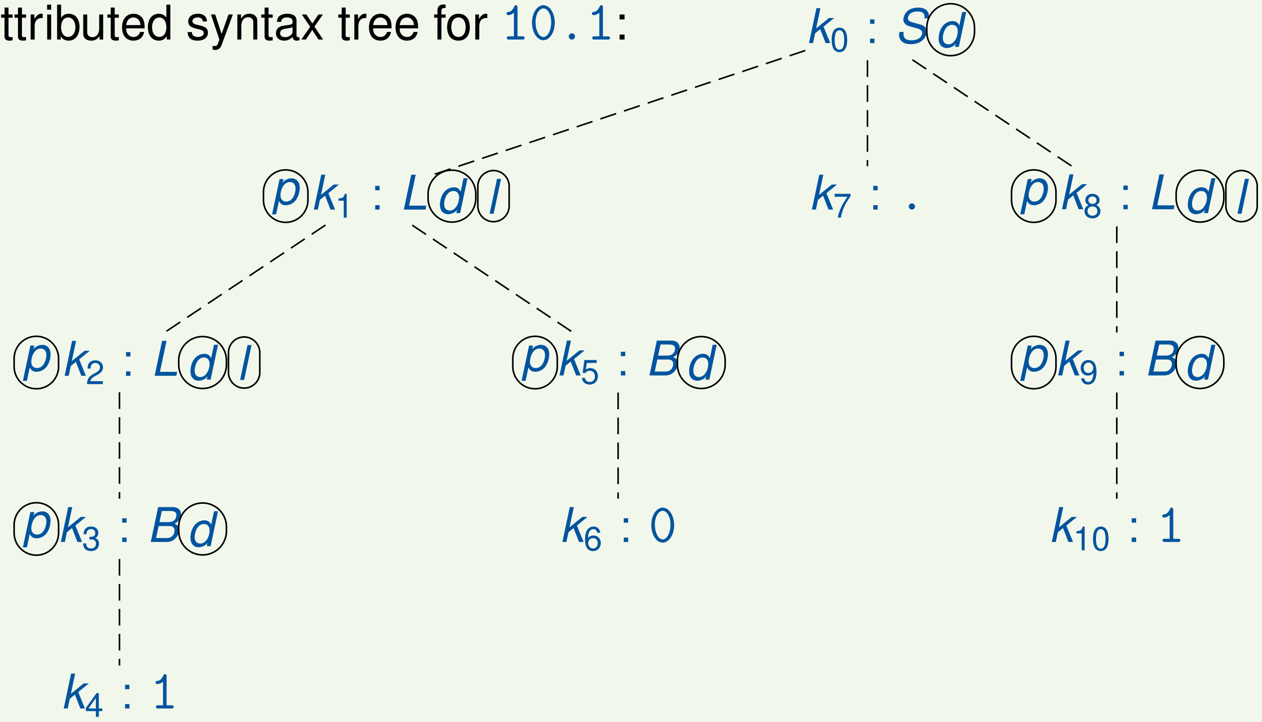
$$E_t := \bigcup \{E_k \mid k \text{ inner node of } t\}.$$

The Attribute Equation System

Attribution of Syntax Trees II

Example 12.6 (cf. Example 12.2)

Attributed syntax tree for `10.1`:

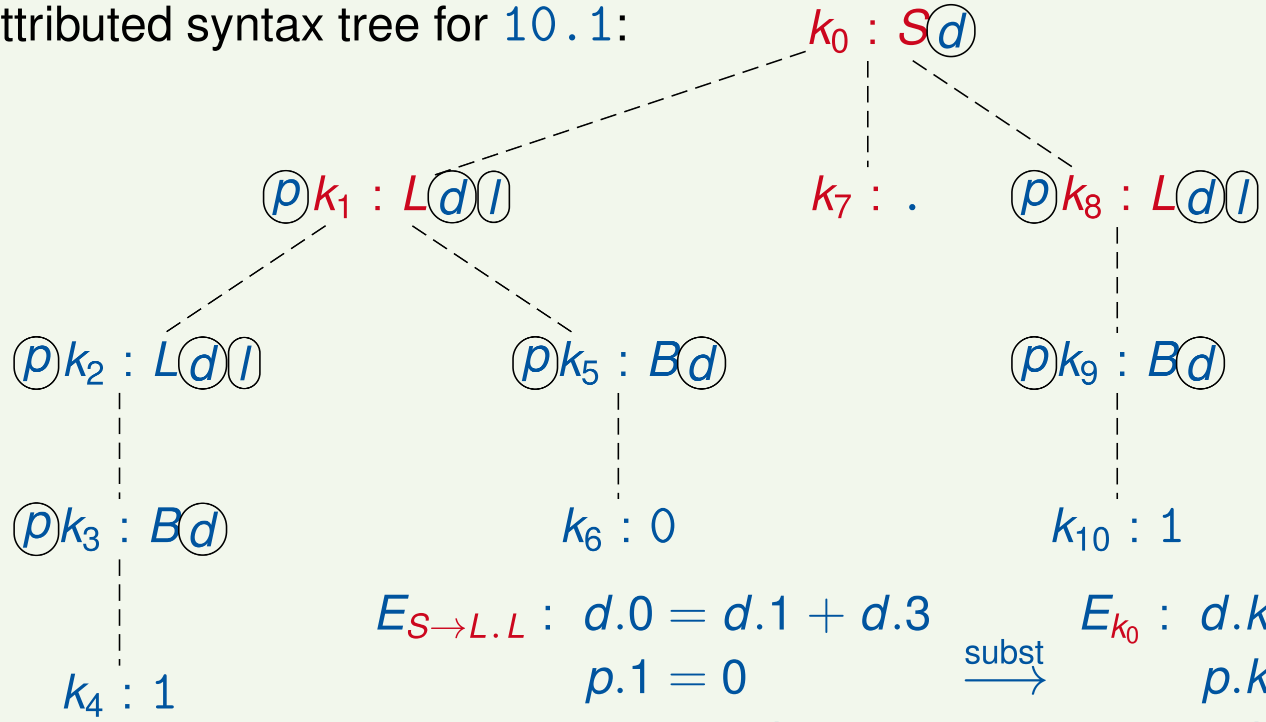


The Attribute Equation System

Attribution of Syntax Trees II

Example 12.6 (cf. Example 12.2)

Attributed syntax tree for `10.1`:



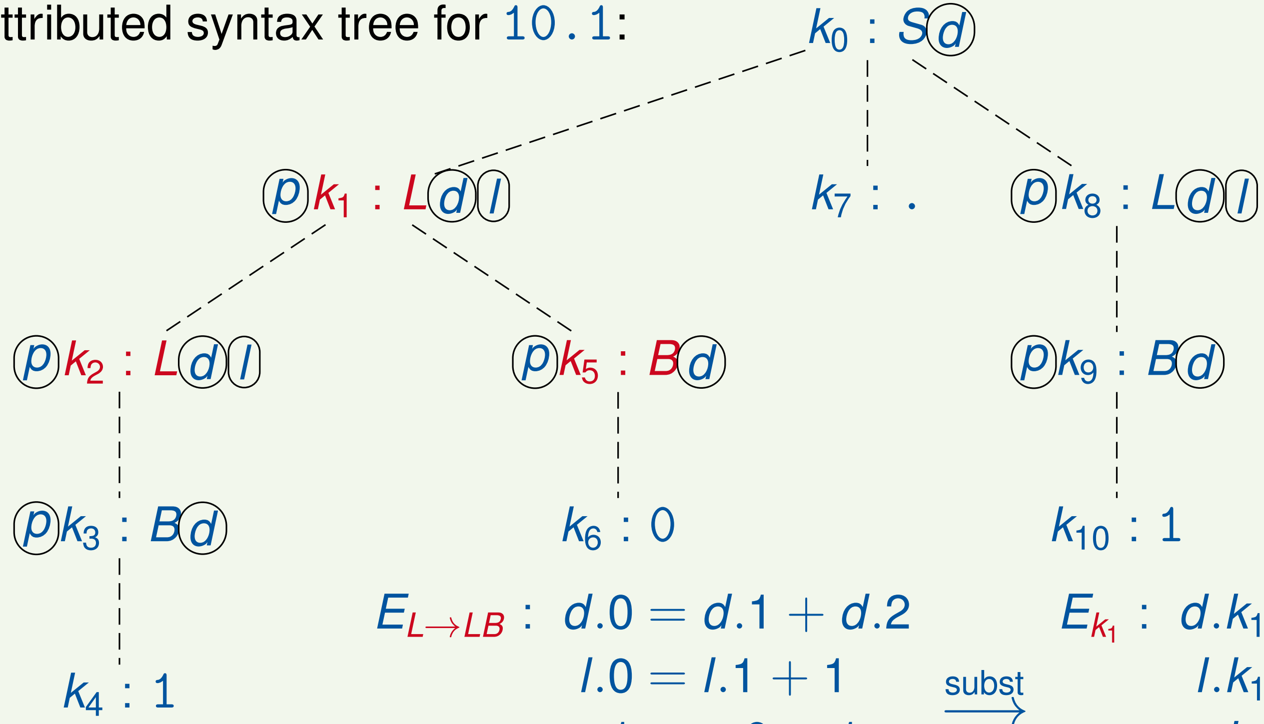
$$
\begin{array}{rl}
E_{S \to L\,.\,L} : & d.0 = d.1 + d.3 \\
& p.1 = 0 \\
& p.3 = -l.3
\end{array}
\xrightarrow{\text{subst}}
\begin{array}{rl}
E_{k_0} : & d.k_0 = d.k_1 + d.k_8 \\
& p.k_1 = 0 \\
& p.k_8 = -l.k_8
\end{array}
$$

The Attribute Equation System

Attribution of Syntax Trees II

Example 12.6 (cf. Example 12.2)

Attributed syntax tree for `10.1`:



$k_0 : S\ d$

$p\ k_1 : L\ d\ l$ $\quad$ $k_7 : .$ $\quad$ $p\ k_8 : L\ d\ l$

$p\ k_2 : L\ d\ l$ $\quad$ $p\ k_5 : B\ d$ $\quad$ $p\ k_9 : B\ d$

$p\ k_3 : B\ d$ $\quad$ $k_6 : 0$ $\quad$ $k_{10} : 1$

$k_4 : 1$

$$E_{L \to LB} : \begin{aligned} d.0 &= d.1 + d.2 \\ l.0 &= l.1 + 1 \\ p.1 &= p.0 + 1 \\ p.2 &= p.0 \end{aligned} \quad \xrightarrow{\text{subst}} \quad E_{k_1} : \begin{aligned} d.k_1 &= d.k_2 + d.k_5 \\ l.k_1 &= l.k_2 + 1 \\ p.k_2 &= p.k_1 + 1 \\ p.k_5 &= p.k_1 \end{aligned}$$

The Attribute Equation System

Attribution of Syntax Trees III

Corollary 12.7

For each $\alpha.k \in Var_t$ 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 $\alpha.k$.

The Attribute Equation System

Attribution of Syntax Trees III

Corollary 12.7

For each $\alpha.k \in Var_t$ *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: $\mathrm{inh}(S) = \emptyset$.
- Synthesized attributes of terminal symbols are provided by the scanner.

Circularity of Attribute Grammars

Outline of Lecture 12

Overview

Semantic Analysis

Attribute Grammars

Adding Inherited Attributes

Formal Definition of Attribute Grammars

The Attribute Equation System

Circularity of Attribute Grammars

Solvability of Attribute Equation System I

Definition 12.8 (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

$$v : Var_t \to V$$

such that, for every $\alpha.k \in Var_t$ and $\alpha.k = f(\alpha.k_1, \ldots, \alpha.k_n) \in E_t$,

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

Circularity of Attribute Grammars

Solvability of Attribute Equation System I

Definition 12.8 (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

$$v : Var_t \to V$$

such that, for every $\alpha.k \in Var_t$ and $\alpha.k = f(\alpha.k_1, \ldots, \alpha.k_n) \in E_t$,

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

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

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

Solvability of Attribute Equation System II

Example 12.9

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

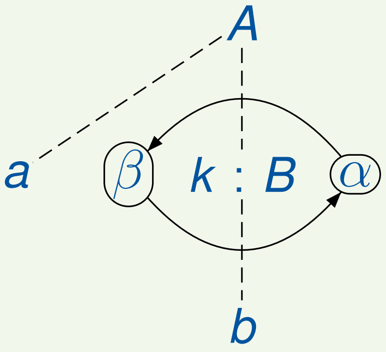
Circularity of Attribute Grammars

Solvability of Attribute Equation System II

Example 12.9

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

$\Longrightarrow$ cyclic dependency:



$$E_t : \ \beta.k = f(\alpha.k)$$
$$\alpha.k = \beta.k$$

Circularity of Attribute Grammars

Solvability of Attribute Equation System II

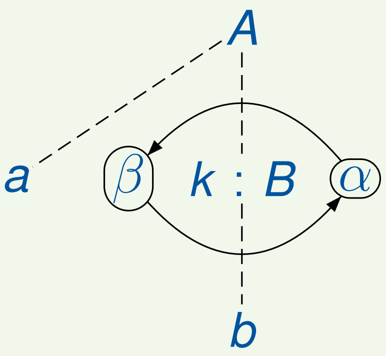
Example 12.9

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

$\Longrightarrow$ for $V^\alpha := V^\beta := \mathbb{N}$ 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 $y \in \mathbb{N})$

$\Longrightarrow$ cyclic dependency:



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

Circularity of Attribute Grammars

Circularity of Attribute Grammars

Goal: unique solvability of equation system

$\Longrightarrow$ avoid cyclic dependencies

Circularity of Attribute Grammars

Circularity of Attribute Grammars

Goal: unique solvability of equation system

$\Longrightarrow$ avoid cyclic dependencies

Definition 12.10 (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

Circularity of Attribute Grammars

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

Definition 12.10 (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.