CS 4300: Compiler Theory

Chapter 5
Syntax-Directed Translation

Dr. Xuejun Liang

Outlines (Sections)

1. Syntax-Directed Definitions
2. Evaluation Orders for SDD's
3. Applications of Syntax-Directed Definition
4. Syntax-Directed Translation Schemes
5. Implementing L-Attributed SDD's

Quick Review of Last Lecture

- Applications of SDD
  - Construction of Syntax Trees During Top-Down Parsing
  - The Structure of a Type

4. Syntax-Directed Translation Schemes

- Any SDT can be implemented by first building a parse tree and then performing the actions in a left-to-right depth-first order.
- Some SDT's can be implemented during parsing, without building a parse tree.
  - But, not all SDT's can be implemented during parsing.
- Focus on the use of SDT's to implement two important classes of SDD's:
  - LR-grammar and S-attributed SDD
  - LL-grammar and L-attributed SDD

Postfix Translation Schemes

SDT's with all actions at the right ends of the production bodies are called postfix SDT's

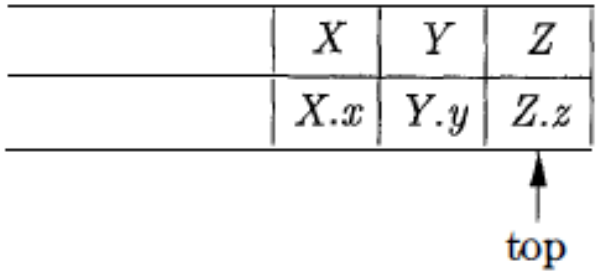
Example: Postfix SDT implementing the desk calculator

$$
\begin{array}{lcll}
L & \rightarrow & E\ \mathbf{n} & \{\ \mathrm{print}(E.val);\ \} \\
E & \rightarrow & E_1 + T & \{\ E.val = E_1.val + T.val;\ \} \\
E & \rightarrow & T & \{\ E.val = T.val;\ \} \\
T & \rightarrow & T_1 * F & \{\ T.val = T_1.val \times F.val;\ \} \\
T & \rightarrow & F & \{\ T.val = F.val;\ \} \\
F & \rightarrow & (\ E\ ) & \{\ F.val = E.val;\ \} \\
F & \rightarrow & \mathbf{digit} & \{\ F.val = \mathbf{digit}.lexval;\ \}
\end{array}
$$

Since the underlying grammar is LR, and the SDD is S-attributed, these actions can be correctly performed along with the reduction steps of the parser

Parser-Stack Implementation of Postfix SDT's

Parser stack



State/grammar symbol

Synthesized attribute(s)

Parser-Stack Implementation of Postfix SDT's

Implementing the desk calculator on a bottom-up parsing stack

PRODUCTION	ACTIONS
$L \rightarrow E\ \mathbf{n}$	$\{\ \text{print}(stack[top-1].val);$ $top = top - 1;\ \}$
$E \rightarrow E_1 + T$	$\{\ stack[top-2].val = stack[top-2].val + stack[top].val;$ $top = top - 2;\ \}$
$E \rightarrow T$	
$T \rightarrow T_1 * F$	$\{\ stack[top-2].val = stack[top-2].val \times stack[top].val;$ $top = top - 2;\ \}$
$T \rightarrow F$	
$F \rightarrow (\ E\ )$	$\{\ stack[top-2].val = stack[top-1].val;$ $top = top - 2;\ \}$
$F \rightarrow \mathbf{digit}$	

SDT's With Actions Inside Productions

- Consider a production: $B \rightarrow X \{\mathbf{a}\} \ Y$
  - The action **a** is done after we have recognized X (if X is a terminal) or all the terminals derived from X (if X is a nonterminal)
- More precisely
  - If the parse is bottom-up, then we perform action **a** as soon as this occurrence of *X* appears on the top of the parsing stack.
  - If the parse is top-down, we perform **a** just before we attempt to expand this occurrence of *Y* (if *Y* is a nonterminal) or check for *Y* on the input (if *Y* is a terminal) .

Problematic SDT for infix-to-prefix translation during parsing

1) $L \rightarrow E\ \mathbf{n}$
2) $E \rightarrow \{\ \text{print}('+');\ \}\ E_1 + T$
3) $E \rightarrow T$
4) $T \rightarrow \{\ \text{print}('*');\ \}\ T_1 * F$
5) $T \rightarrow F$
6) $F \rightarrow (\ E\ )$
7) $F \rightarrow \mathbf{digit}\ \{\ \text{print}(\mathbf{digit}.\mathit{lexval});\ \}$

- Unfortunately, it is impossible to implement this SDT during either top-down or bottom-up parsing, because the parser would have to perform critical actions, like printing instances of * or +, long before it knows whether these symbols will appear in its input

SDT's With Actions Inside Productions

- Consider a production: $B \rightarrow X \{\mathbf{a}\} Y$
  - The action **a** is done after we have recognized X (if X is a terminal) or all the terminals derived from X (if X is a nonterminal)
- Insert marker nonterminals to remove the embedded action and to change the SDT to a postfix SDT
  - Rewrite the product with marker nonterminal M into

$$B \rightarrow X\, M\, Y$$
$$M \rightarrow \varepsilon \{\mathrm{a}\}$$

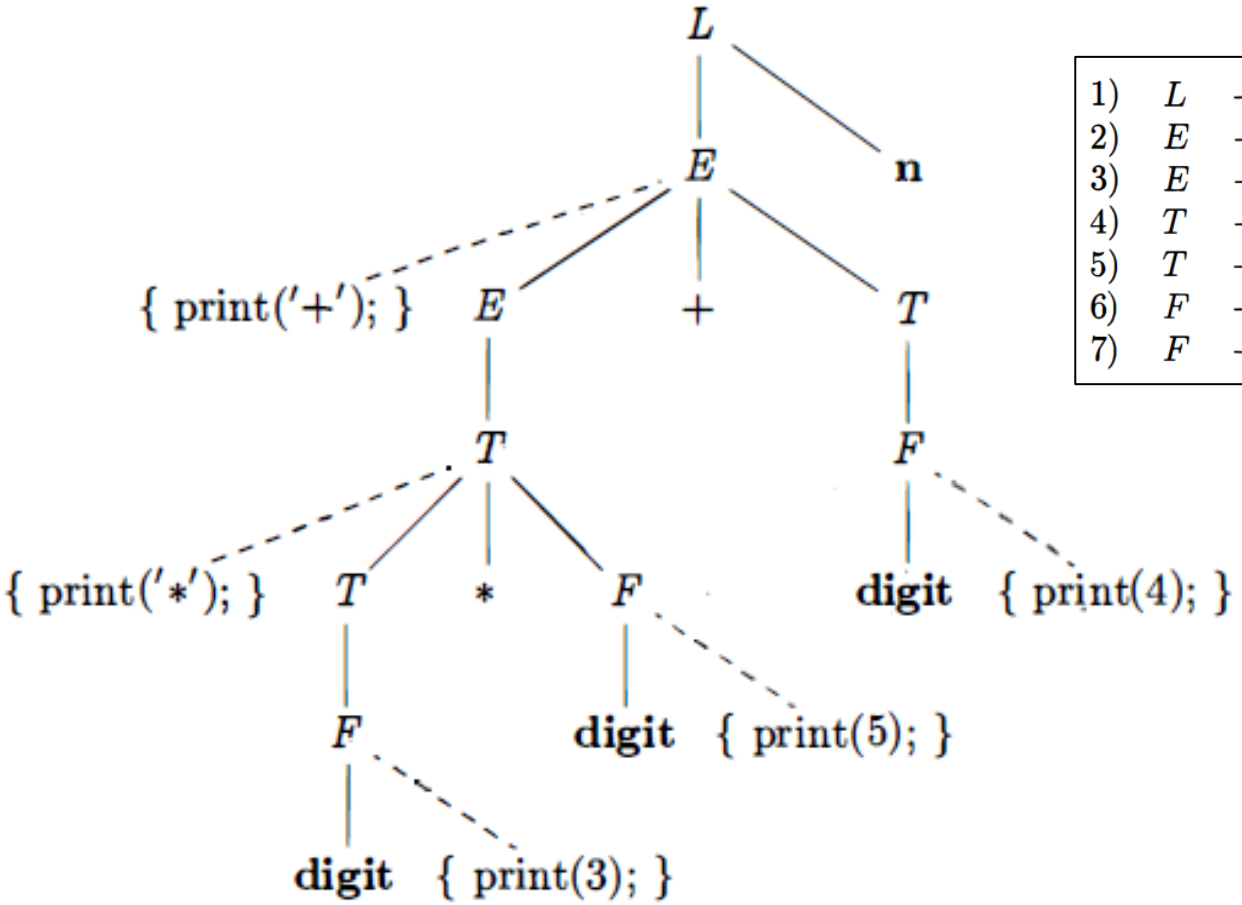
- Problems with inserting marker nonterminals
  - May introduce conflicts in the parse table
  - How to propagate inherited attributes?

Any SDT Can Be Implemented

1. Ignoring the actions, parse the input and produce a parse tree as a result.
2. Then, examine each interior node N, say one for production $A \rightarrow \alpha$. Add additional children to N for the actions in α, so the children of N from left to right have exactly the symbols and actions of α.
3. Perform a preorder traversal of the tree, and as soon as a node labeled by an action is visited, perform that action.

Parse Tree With Actions Embedded

3 * 5 + 4 ⟹ + * 3 5 4

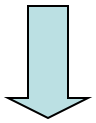


1)	L	$\rightarrow$	E **n**
2)	E	$\rightarrow$	{ print('+'); } $E_1 + T$
3)	E	$\rightarrow$	T
4)	T	$\rightarrow$	{ print('*'); } $T_1 * F$
5)	T	$\rightarrow$	F
6)	F	$\rightarrow$	(E)
7)	F	$\rightarrow$	**digit** { print(**digit**.*lexval*); }

Eliminating Left Recursion From SDT's

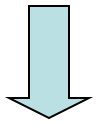
When the order in which the actions in an SDT is needed to consider only, the actions are treated as if they were terminal symbols during transforming the grammar,

$A \rightarrow A\ \alpha$
$A \rightarrow \beta$



$A \rightarrow \beta\ R$
$R \rightarrow \alpha\ R$
$R \rightarrow \varepsilon$

$E \rightarrow E + T$ {print('+');}
$E \rightarrow T$



$E \rightarrow T\ R$
$R \rightarrow + T$ {print('+');} R
$R \rightarrow \varepsilon$

Eliminating Left Recursion (Cont.)

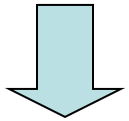
Consider the following *SDD* with a single recursive production, single nonrecursive production, and a single attribute of the left-recursive nonterminal.

$$A \rightarrow A_1\ Y\ \{\ A.\mathrm{a} = g(A_1.\mathrm{a},\ Y.\mathrm{y})\ \}$$
$$A \rightarrow X\ \{\ A.\mathrm{a} = f(X.\mathrm{x})\ \}$$

$$A \rightarrow X\ \{\ R.\mathrm{i} = f(X.\mathrm{x})\ \}\ R\ \{\ A.\mathrm{a} = R.\mathrm{s}\ \}$$
$$R \rightarrow Y\ \{\ R_1.\mathrm{i} = g(R.\mathrm{i},\ Y.\mathrm{y})\ \}\ R_1\ \{\ R.\mathrm{s} = R_1.\mathrm{s}\ \}$$
$$R \rightarrow \varepsilon\ \{\ R.\mathrm{s} = R.\mathrm{i}\ \}$$

Underlining grammar

$$A \rightarrow A\ Y$$
$$A \rightarrow X$$

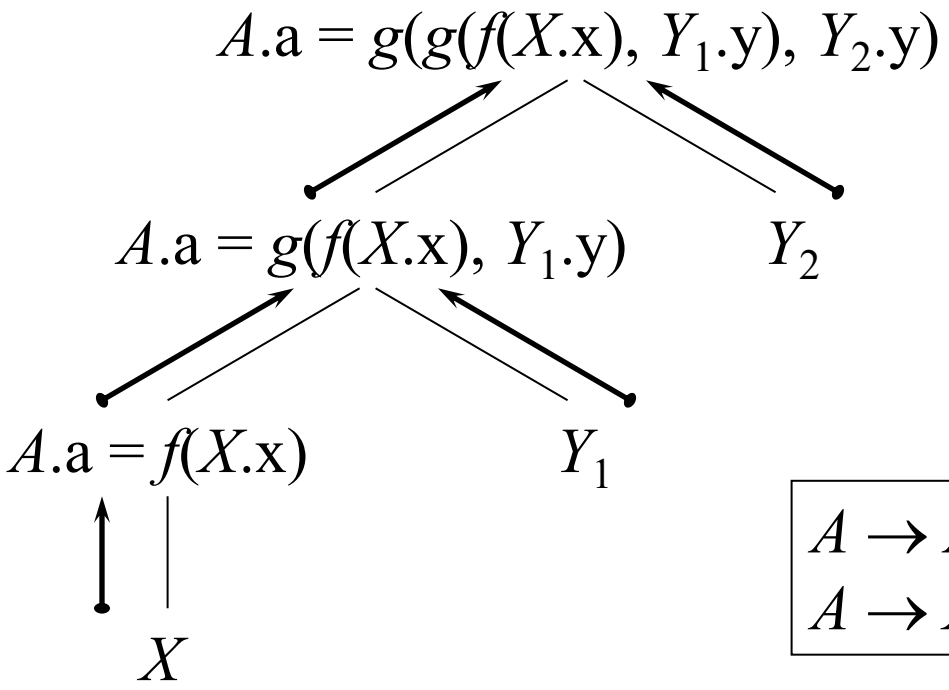


$$A \rightarrow X\ R$$
$$R \rightarrow Y\ R$$
$$R \rightarrow \varepsilon$$

Eliminating Left Recursion (Cont.)

Consider input string $XYY = XY_1Y_2$

$$A \rightarrow A\ Y$$
$$A \rightarrow X$$

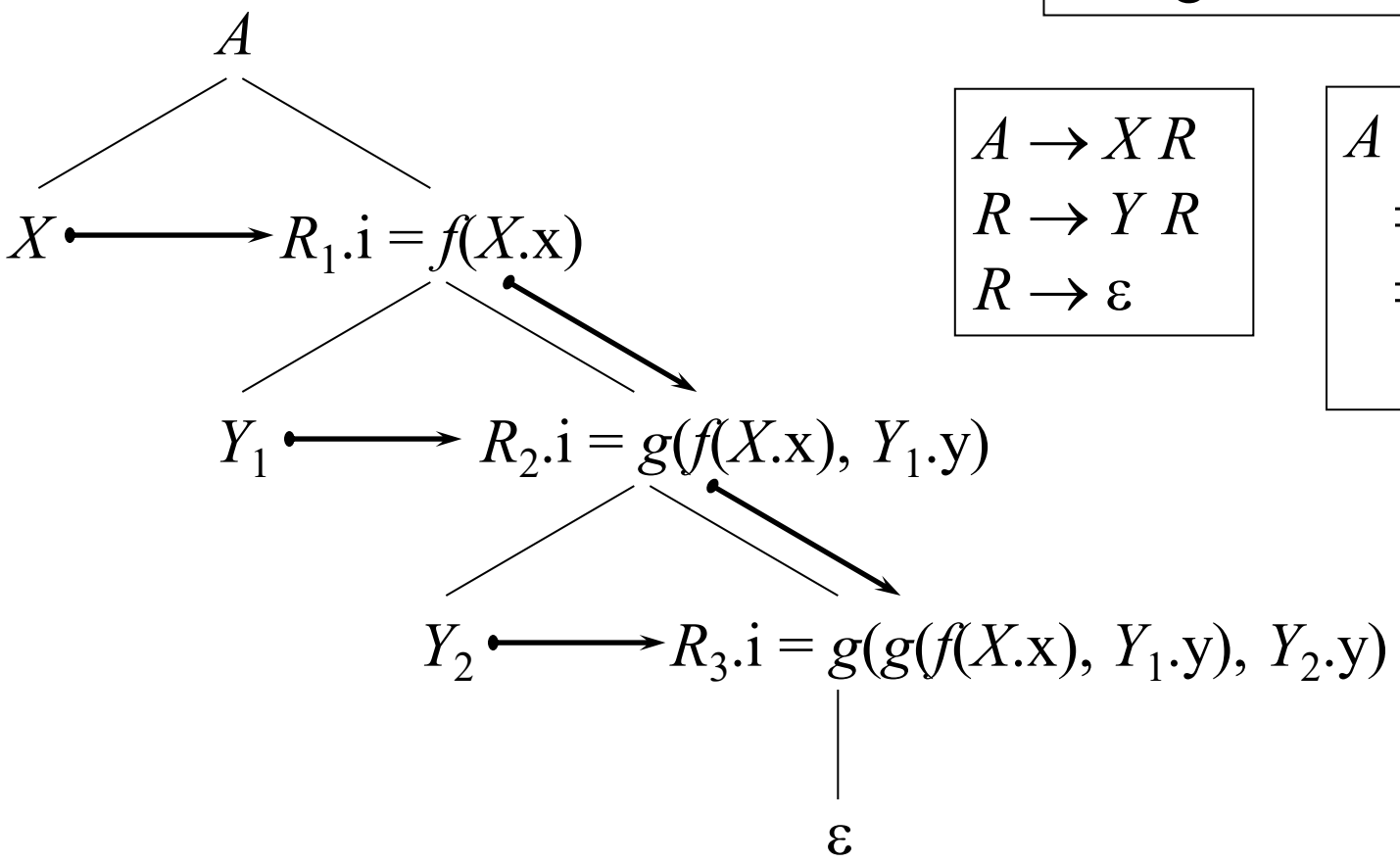


$$A \Rightarrow A\ Y_2$$
$$\Rightarrow A\ Y_1\ Y_2$$
$$\Rightarrow X\ Y_1\ Y_2$$

$$A \rightarrow A_1\ Y \ \{\ A.a = g(A_1.a,\ Y.y)\ \}$$
$$A \rightarrow X \ \{\ A.a = f(X.x)\ \}$$

1. Flow of inherited attribute values

Consider the same input string $XYY = XY_1Y_2$



$A \to X\,R$
$R \to Y\,R$
$R \to \varepsilon$

$A \Rightarrow X\,R$
$\Rightarrow X\,Y_1\,R$
$\Rightarrow X\,Y_1\,Y_2\,R$
$\Rightarrow X\,Y_1\,Y_2$

$A \to A_1\,Y\ \{\ A.\mathrm{a} = g(A_1.\mathrm{a},\ Y.\mathrm{y})\ \}$
$A \to X\ \{\ A.\mathrm{a} = f(X.\mathrm{x})\ \}$

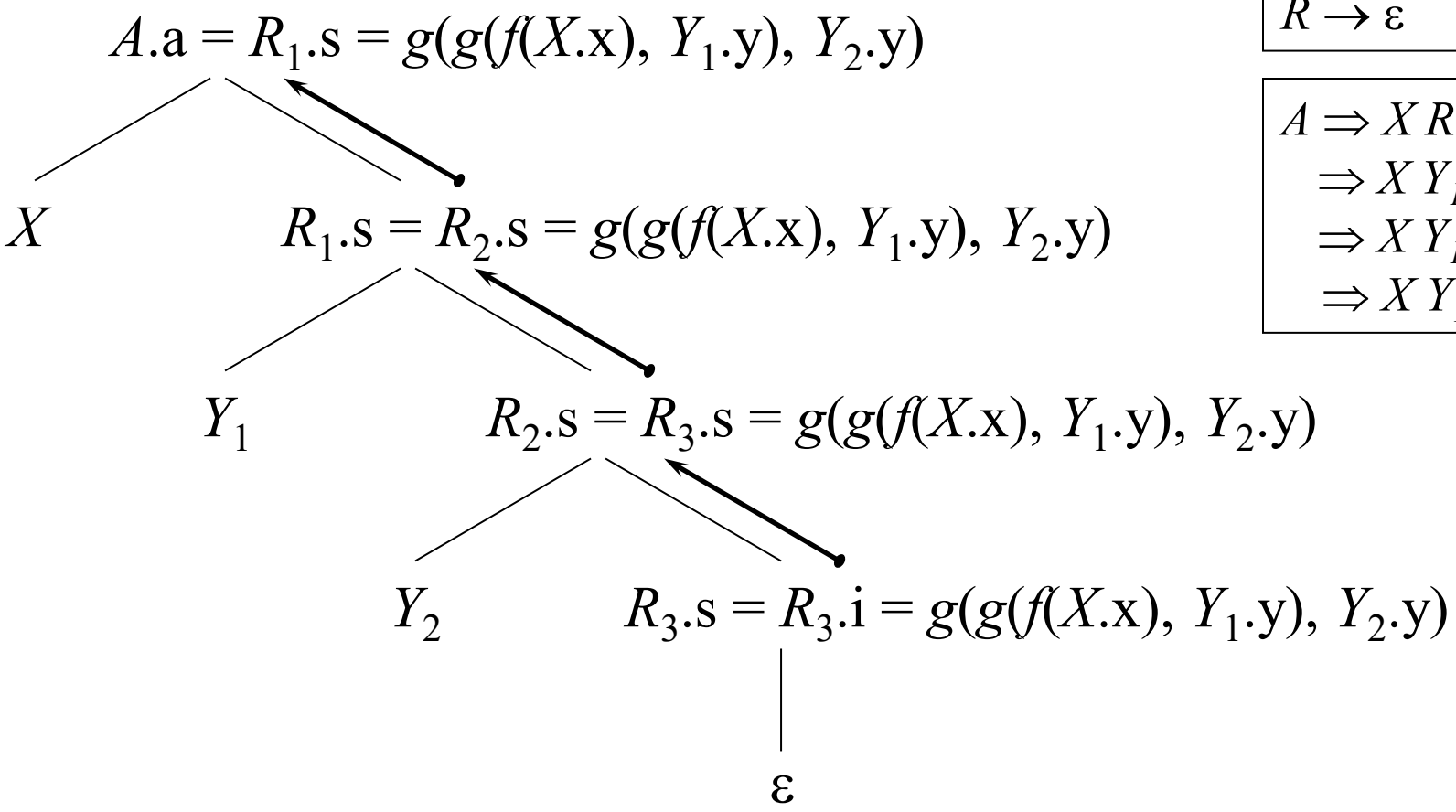
$\Rightarrow$

$A \to X\ \{\ R.\mathrm{i} = f(X.\mathrm{x})\ \}\ R\ \{\ A.\mathrm{a} = R.\mathrm{s}\ \}$
$R \to Y\ \{\ R_1.\mathrm{i} = g(R.\mathrm{i},\ Y.\mathrm{y})\ \}\ R_1\ \{\ R.\mathrm{s} = R_1.\mathrm{s}\ \}$
$R \to \varepsilon\ \{\ R.\mathrm{s} = R.\mathrm{i}\ \}$

2. Flow of synthesized attribute values

$A \to X\ R$
$R \to Y\quad R$
$R \to \varepsilon$

$A \Rightarrow X\ R$
$\Rightarrow X\ Y_1\ R$
$\Rightarrow X\ Y_1\ Y_2\ R$
$\Rightarrow X\ Y_1\ Y_2$



$A \to A_1\ Y\ \{\ A.\mathrm{a} = g(A_1.\mathrm{a},\ Y.\mathrm{y})\ \}$
$A \to X\ \{\ A.\mathrm{a} = f(X.\mathrm{x})\ \}$

$\Rightarrow$

$A \to X\ \{\ R.\mathrm{i} = f(X.\mathrm{x})\ \}\ R\ \{\ A.\mathrm{a} = R.\mathrm{s}\ \}$
$R \to Y\ \{\ R_1.\mathrm{i} = g(R.\mathrm{i},\ Y.\mathrm{y})\ \}\ R_1\ \{\ R.\mathrm{s} = R_1.\mathrm{s}\ \}$
$R \to \varepsilon\ \{\ R.\mathrm{s} = R.\mathrm{i}\ \}$

SDT's for L-Attributed Definitions

- Assume that the underlying grammar can be parsed top-down
- The rules for turning an L-attributed SDD into an SDT are as follows
  1. Embed the action that computes the inherited attributes for a nonterminal A immediately before that occurrence of A in the body of the production. If several inherited attributes for A depend on one another in an acyclic fashion, order the evaluation of attributes so that those needed first are computed first.
  2. Place the actions that compute a synthesized attribute for the head of a production at the end of the body of that production.