



INF 5110: Compiler construction

Spring 2023

Series 5

27. 3. 2023

Topic: Chapter 6: Attribute grammars

Issued: 27. 3. 2023

Exercise 1 (Post-fix printout) Rewrite the attribute grammar shown below to compute a *postfix* string attribute instead of a value *val*, containing the postfix form for the simple integer expression.¹ For example, the postfix attribute for

$$(34 - 3) * 42 \text{ is } "34 3 - 42 *"$$

You may assume a string concatenation operator `||` and the existence of a `number.strval` attribute.²

	productions/grammar rules	semantic rules
1	$exp_1 \rightarrow exp_2 + term$	$exp_1.val = exp_2.val + term.val$
2	$exp_1 \rightarrow exp_2 - term$	$exp_1.val = exp_2.val - term.val$
3	$exp \rightarrow term$	$exp.val = term.val$
4	$term_1 \rightarrow term_2 * factor$	$term_1.val = term_2.val * factor.val$
5	$term \rightarrow factor$	$term.val = factor.val$
6	$factor \rightarrow (exp)$	$factor.val = exp.val$
7	$factor \rightarrow \mathbf{number}$	$factor.val = \mathbf{number.val}$

Table 1: AG for evaluation (from the lecture)

□

Exercise 2 (Simple typing via AGs) Consider the following *grammar* for simple Pascal-style declarations.

¹As a preview for one of the later chapters: in the context of *intermediate code generation*, we will cover a specific form of intermediate code, so called *p-code* (or one address code, etc.) *Generating* intermediate p-code from ASTs resembles the task at hand, in that code generation there involves post-fix emission of lines of code, at least for straight-line code involving expressions. You may also be reminded of the “AST-pretty-printer” of the oblig: one recommended form of output was basically a *prefix*-printout of the tree (maybe indented for easier human consumption).

²Postfix notation is otherwise also known as *reverse polish notation*, which is actually predates modern electronic computers (at least the non-reversed Polish notation), but has been kind of popular in certain pocket calculators (especially Hewlett-Packard). Also in the context of depth-first tree traversal, there is pre-fix/post-fix/in-order treatment of nodes of the traversal, which is related to the task here, as well.

$$\begin{aligned}
 \text{decl} &\rightarrow \text{var-list} : \text{type} \\
 \text{var-list} &\rightarrow \text{var-list} , \text{id} \mid \text{id} \\
 \text{type} &\rightarrow \mathbf{integer} \mid \mathbf{real}
 \end{aligned}$$

Write an attribute grammar for the *type* of a variable.

Exercise 3 (Dependency graphs and evaluation) Consider the following attribute grammar.

productions/grammar rules	semantic rules
$S \rightarrow ABC$	$B.u = S.u$ $A.u = B.v + C.v$ $S.v = A.v$
$A \rightarrow a$	$A.v = 2 * A.u$
$B \rightarrow b$	$B.v = B.u$
$C \rightarrow c$	$C.v = 1$

1. Draw the parse tree for the string **abc** (the only word in the language) and draw the dependency graph for the associated attributes. Describe a correct order for the evaluation of the attributes.
2. Suppose that the value 3 is assigned to $S.u$ before attribute evaluation begins. What is the value of $S.v$ when the evaluation has finished.
3. Suppose the attribute equations are modified as follows:

production/grammar rule	semantic rules
$S \rightarrow ABC$	$B.u = S.u$ $C.u = A.v$ $A.u = B.v + C.v$ $S.v = A.v$
$A \rightarrow a$	$A.v = 2 * A.u$
$B \rightarrow b$	$B.v = B.u$
$C \rightarrow c$	$C.v = C.u - 2$

What value does $S.v$ have after attribute evaluation, if $S.u = 3$ before the evaluation begins?

Exercise 4 (AG for classes) Consider the following grammar for class declarations:

$$\begin{aligned}
 \text{class} &\rightarrow \mathbf{class} \text{ name } \text{superclass} \{ \text{decls} \} \\
 \text{decls} &\rightarrow \text{decls} ; \text{decl} \mid \text{decl} \\
 \text{decl} &\rightarrow \text{variable-decl} \\
 \text{decl} &\rightarrow \text{method-decl} \\
 \text{method-decl} &\rightarrow \text{type} \text{ name } (\text{params}) \text{ body} \\
 \text{type} &\rightarrow \mathbf{int} \mid \mathbf{bool} \mid \mathbf{void} \\
 \text{superclass} &\rightarrow \mathbf{name}
 \end{aligned}$$

As usual, terminals are indicated in boldface, where for **name**, we assume that it represents names the scanner provides; **name** is assumed to have an attribute **name**.

Methods with the same name as the class they belong to are *constructor methods*. For those, the following informal typing “rule” is given:

Constructors need to be specified with the type **void**.

Design semantical rules for this requirement for the following fragment of an AG.

productions/grammar rules	semantic rules
<i>class</i> → class name <i>superclass</i> { <i>decls</i> }	
<i>decls</i> → <i>decls</i> ; <i>decl</i>	
<i>decls</i> → <i>decl</i>	
<i>decl</i> → <i>variable-decl</i>	not to be filled out
<i>decl</i> → <i>method-decl</i>	
<i>method-decl</i> → <i>type</i> name (<i>params</i>) <i>body</i>	
<i>type</i> → int	
<i>type</i> → bool	
<i>type</i> → void	
(<i>superclass</i> → name)	filled by lexer