Universitetet i Oslo
Institutt for Informatikk

Reliable Systems
Martin Steffen, Gianluca Turin

INF 5110: Compiler construction

Spring 2021

Series 5

12. 3. 2021

Topic: Chapter 6: Attribute grammars

Issued: 12. 3. 2021

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

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

You may assume a string concatenation operator $\|$ and the existence of a **number**.`strval` attribute.[2]

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

Table 1: AG for evaluation (from the lecture)

□

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

[1] As 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).

[2] Postfix 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.

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

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
$S \to ABC$	$B.\mathtt{u} = S.\mathtt{u}$
	$A.\mathtt{u} = B.\mathtt{v} + C.\mathtt{v}$
	$S.\mathtt{v} = A.\mathtt{v}$
$A \to a$	$A.\mathtt{v} = 2 * A.\mathtt{u}$
$B \to b$	$B.\mathtt{v} = B.\mathtt{u}$
$C \to c$	$C.\mathtt{v} = 1$

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 $S.\mathtt{u}$ before attribute evaluation begins. What is the value of $S.\mathtt{v}$ when the evaluation has finished.

3. Suppose the attribute equations are modified as follows:

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

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

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

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

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 atrribute `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
class	→	**class name** *superclass* **{** *decls* **}**	
decls	→	*decls* **;** *decl*	
decls	→	*decl*	
decl	→	*variable-decl*	not to be filled out
decl	→	*method-decl*	
method-decl	→	*type* **name** **(** *params* **)** *body*	
type	→	**int**	
type	→	**bool**	
type	→	**void**	
(*superclass*	→	**name**)	filled by lexer