7CS5A COMPILER CONSTRUCTION (Syntax Directed Translation) Fall 2018

Lecture 2: Sept. 28, 2018

Instructor: K.R. Chowdhary *: Professor of CS*

2.1 Syntax Directed Translation

Syntax-directed translation is done by attaching rules or program fragments to productions in a grammar. For example, consider an expression *expr* generated by the production,

$$expr \rightarrow expr_1 + term.$$

The subscript in $expr_1$, is used only to distinguish the instance of *expr* in the production body from the head of the production. We can translate *expr* by exploiting its structure, as in the following pseudo-code:

$$\begin{aligned}&translate\ expr_1;\\&translate\ term;\\&handle\ the\ +;\end{aligned} \tag{2.1}$$

In addition to program fragments, we associate information with a language construct by attaching attributes to the grammar symbol(s) representing the construct. A syntax-directed definition (SDD) specifies the values of attributes by embedding semantic rules in the grammar productions. For example, an infix-to-postfix translator might have a production and rule:

$$Production : E \rightarrow E_1 + T$$

$$\begin{aligned}&Corresponding\ Semantic\ Rule :\\&E.code = E_1.code \parallel T.code \parallel '+'\end{aligned}$$

This production has two nonterminals, E and T; the subscript in E_1 distinguishes the occurrence of E in the production body from the occurrence of E as the head. Both E and T have a string-valued attribute code[1]. The semantic rule specifies that the string

[1]The word *code* here is for identification, and not for program code. This attribute code supports concatenation.

$E.code$ is formed by concatenating ($\|$ is symbol for concatenation) $E_1.code$, $T.code$, and the character $'+'$. While the rule makes it explicit that the translation of E is built up from the translations of E_1, T, and $'+'$, it may be inefficient to implement the translation directly by manipulating strings, for example, we may need to do type change to perform arithmetic.

A syntax-directed translation (SDT) scheme embeds program fragments called *semantic actions* within production bodies, as follows:

$$E \to E_1 + T \quad \{print'+'\} \tag{2.2}$$

By convention, *semantic actions* are enclosed within curly braces. (If curly braces occur as grammar symbols, we enclose them within single quotes, as in '{' and '}'.) The position of a semantic action in a production body determines the order in which the action is executed (e.g., in preorder, post order, and inorder). In production (2.2), the action occurs at the end, after all the grammar symbols. In general, a semantic actions may occur at any position in a production body.

Between the two notations, syntax-directed definitions can be more readable, and hence more useful for specifications. However, translation schemes can be more efficient, and hence more useful for implementations.

The most general approach to syntax-directed translation is to construct a parse tree or a syntax tree, and then to compute the values of attributes at the nodes of the tree by visiting the nodes of the tree. In other cases, translation can be done during parsing, without building an explicit tree. We shall therefore study a class of syntax-directed translations called "L-attributed translations" (L for left-to-right), which covers virtually all translations that can be performed during parsing. We also study a smaller class, called "S-attributed translations" (S for synthesized), which can be performed easily in connection with a bottom-up parse.

There are two important concepts related to *syntax directed translation*: 1. Attribute, and 2. syntax-directed translation schema.

Attributes. An attribute is any quantity associated with a programming construct, for example, data types in the expressions, the number of instructions in the generated code, or the location of the first instruction in the generated code, etc. In the programming constructs, we use use grammar symbols (nonterminals and terminals), we extend the notion of attributes from constructs to the symbols to represent them.

Syntax-directed translation schema. A translation scheme is a notation for attaching program fragments to the productions of a grammar. These program fragments are executed when the production rule is applied during *syntax analysis.* The combined result of all these fragment executions, in the order of the syntax analysis, produces the translation of the program to which this analysis/synthesis process is applied.

We will use Syntax-directed translations to translate infix expressions into postfix notation, to evaluate expressions, and to build syntax trees for programming constructs.

2.1.1 Postfix Notation

We will discuss translation of expressions into postfix notation. A postfix notation for an expression E can be defined *inductively* as follows:

1. If E is a variable or constant, then the postfix notation for E is E itself.
2. If E is an expression of the form E_1 *op* E_2, where *op* is an arbitrary operator, then postfix notation for E is "$E'_1\ E'_2\ \ op$", where E'_1 and E'_2 are the postfix notations for E_1 and E_2, respectively. The E_1 and E'_2, will not be same unless they are constants or variables. Similar is case with E_2 vs E'_2.
3. If E is a parenthesized expression of the form (E_1), then the postfix notation for E is the same as the postfix notation for E_1.

Example 2.1 *Translating to posfix notation.*

The postfix notation for $(7 - 3) + 5$ is $7\ 3 - 5+$. The translation of 7, 3, 5 are constants, by rule (1). Then the translation of $7 - 3$ is $7\ 3-$ by rule (2). The translation of $(7 - 3)$ is same by rule (3) above.

Having translated the parenthesized subexpression, we apply the rule (2), to the entire expression, with $(7 - 3)$ in the role of E_1 and 5 in the role of E_2, to get result $7\ 3 - 5+$.

As another example, the postfix notation for $7 - (3 + 2)$ is $7\ 3\ 2 + -$. That is $3 + 2$ is first translated into $3\ 2+$, which become the second argument of minus sign. □

No parentheses are needed in postfix notation, because the position and arity (number of arguments) of the operators permits only one possible decoding of a postfix expression. The "trick" to compute postfix expression is to repeatedly scan the postfix string from the left most, until you find an operator. Then, look to the left for the proper number of operands, and group this operator with its operands. Then evaluate the operator on the operands, and replace them by the result. Then repeat the process, continuing to the right and searching for another operator.

Consider the postfix expression $952 + -3*$, we find that it evaluates as follows:

$$\begin{aligned} 9\ 5\ 2\ +\ \ -\ \ 3 * &\Rightarrow 9\ 7\ -\ 3 * \\ &\Rightarrow 2\ 3 * \\ &\Rightarrow 6. \end{aligned}$$

2.1.2 Synthesized Attributes and annotated Parse Tree

It is idea of associating quantities with programming constructs–for example, values and types with expressions–can be expressed in terms of grammars. We associate attributes with nonterminals and terminals. Then, we attach rules to the productions of the grammar; these rules describe how the attributes are computed at those nodes of the parse-tree where the production in question is used to relate a node to its children. For example, $x \to 5{-}7$, will represent a subtree with x as root and $5,' -', 7$ as children. We should associate attributes with these so that these attributes help for correctly computing values at root node x.

A syntax-directed definition (SDD) associates:

1. With each grammar symbol, a set of attributes, and
2. With each production, a set of semantic rules for computing the values of the attributes associated with the symbols appearing in the production.

Attributes can be evaluated as follows: For a given input string x, construct a parse tree for x. Then, apply the semantic rules to evaluate attributes at each node in the parse tree. Suppose a node N in a parse tree is labeled by the grammar symbol X . We write $X.a$ to denote the value of attribute a of X at that node.

Definition 2.2 ***Synthesized Attribute.*** *An attribute is said to be synthesized if its value at a parse-tree node N is determined from attribute values at the children of N and at N itself.* □

Synthesized attributes have the desirable property that they can be evaluated during a single bottom-up traversal of a parse tree.

Example 2.3 *Syntax-directed definition for infix to postfix translation.*

The annotated parse tree in Fig. 2.1 is based on the syntax-directed definition in Table 2.1 for translating expressions consisting of digits separated by plus or minus signs into postfix notation. The attribute t indicates the attribute as postfix of *term* and *expr*. Each nonterminal has a string-valued attribute t that represents the postfix notation for the expression generated by that nonterminal in a parse tree. The symbol $\|$ in the semantic rule is the operator for string concatenation.

Table 2.1: Syntax-directed definition for infix to postfix translation.

Production	Semantic Rules
$expr \rightarrow expr_1 + term$	$expr.t = expr_1.t \parallel term.t \parallel {'+'}$
$expr \rightarrow expr_1 - term$	$expr.t = expr_1.t \parallel term.t \parallel {'-'}$
$expr \rightarrow term$	$expr.t = term.t$
$term \rightarrow 0$	$term.t =' 0'$
$term \rightarrow 1$	$term.t =' 1'$
...	...
$term \rightarrow 9$	$term.t =' 9'$

The postfix form of a digit is the digit itself; e.g., the semantic rule associated with the production $term \rightarrow 7$ defines $term.t$ to be 7 itself whenever this production is used at a node in a parse tree, and similar action is taken for other digits. As another example, when the production $expr \rightarrow term$ is applied, the value of $term.t$ becomes the value of $expr.t$ ($term.t$'s value moves up into value of $expr.t$), in the left most leg of the tree in figure 2.1.

The production $expr \rightarrow expr_1 + term$ derives an expression containing a plus operator. The left operand of the plus operator is given by $expr_1$, and the right operand by *term*. The semantic rule

$$exper.t = expr_1.t \parallel term.t \parallel {'+'}$$

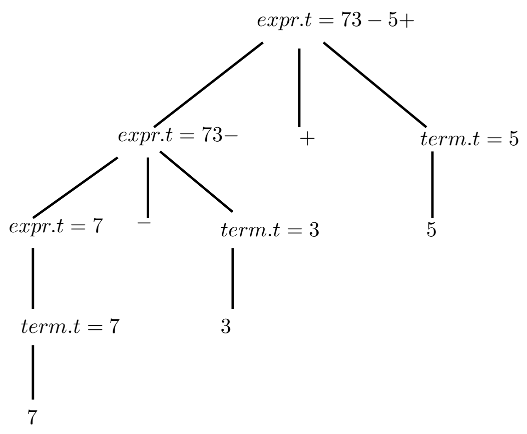

Figure 2.1: Annotated Parse Tree for $7-3+5$.

associated with this production constructs the value of attribute *expr.t* by concatenating the postfix forms $expr_1.t$ and *term.t* of the left and right operands, respectively, and then appending the plus sign. This rule is a formalization of the definition of "postfix expression." □

Definition 2.4 ***Annotated parse tree.*** *A parse tree showing the attribute values at each node is called an* annotated parse tree. □

For example, Fig. 2.1 shows an annotated parse tree for expression $7-3+5$ with an attribute t associated with the nonterminals *expr* and *term*. The value $73-5+$ of the attribute at the root node, which is the postfix notation for $7-3+5$. We shall see shortly how these expressions are computed.

Definition 2.5 Simple syntax-directed Translation. *The syntax-directed definition in Example 2.3 has the following important property: the string representing the translation of the nonterminal at the head of each production is the concatenation of the translations of the nonterminals in the production body, in the same order as in the production, with some optional additional strings interleaved.* □

Simple syntax-directed definition can be implemented by printing only the additional strings, in the order they appear in the definition.

2.2 Exercises

1. Convert the expression $(3+5)/4*2$ into postfix notation. Write all the necessary steps for this.
2. Give an inductive definition of obtaining postfix expression. Why the definition given in this chapter is not called as recursive definition.
3. Write an algorithm in your own language to convert any general expression into postfix notation.

4. Write an algorithm in your own language to evaluate any given expression of postfix expression.

5. Demonstrate computing a postfix expression obtained from $(3+5)/4*2$.

6. Construct an annotated parse tree for $3*5+7$.

7. (a) When an attributed is called synthesized attribute?
   (b) What is special property of a synthesized attribute?
   (c) What can be the semantic actions, other than *print*? Give examples.
   (d) Why parentheses are not required in the postfix expression? Justify.

8. Define followings:
   (a) Syntax Directed Definition
   (b) Simple Syntax Directed translation
   (c) Annotated Parse Tree

9. Construct annotated parse trees for each of the following.
   (a) $3*2-8+5$
   (b) $5/2+7-6$
   (c) $8*3+6/2$