7CS5A COMPILER CONSTRUCTION (Syntax Directed Translation) Fall 2018

Lecture 3: Sept. 29, 2018

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

3.1 Tree Traversals

Tree traversals will be used for describing attribute evaluation and for specifying the execution of code fragments in a translation scheme. A traversal of a tree starts at the root and visits each node of the tree in some order.

A depth-first traversal starts at the root and recursively visits the children of each node in any order, not necessarily from left to right. It is called "depth- first" because it visits an "unvisited" child of a node whenever it can, so it visits nodes as far away from the root (as "deep") as quickly as it can.

The procedure *visit*(N) in Algorithm 1 is a depth first traversal that visits the children of a node in left-to-right order. In this traversal, we have included the action of evaluating translations at each node, just before we finish with that node (i.e., after translations at the children have surely been computed). In general, the actions associated with a traversal can be whatever we choose, or nothing at all.

Algorithm 1 visit(N)

```
1: Procedure visit(node N){
2: for (each child C of N, from left to right) do
3:    visit(C);
4: end for
5: evaluate semantic rules at node N;
6: }
```

A syntax-directed definition does not impose any specific order for the evaluation of attributes on a parse tree; any evaluation order that computes an attribute a after all the other attributes that a depends on is acceptable. Synthesized attributes can be evaluated during any bottom-up traversal, that is, a traversal that evaluates attributes at a node after having evaluated attributes at its children. In general, after considering the both *synthesized* and *inherited* attributes, the matter of evaluation order is quite complex.

3.2 Translation Schemes

The syntax-directed definition we discussed earlier (for infix to postfix translation) builds up a translation by attaching strings as attributes to the nodes in the parse tree. We now consider an alternative approach that does not need to manipulate strings; it produces the same translation increment ally, by executing program fragments.

A syntax-directed translation scheme is a notation for specifying a translation by attaching program fragments to productions in a grammar. A translation scheme is like a syntax-directed definition, except that the order of evaluation of the semantic rules is explicitly specified.

Program fragments embedded within production bodies are called *semantic actions*. The position at which an action is to be executed is shown by enclosing it between curly braces and writing it within the production body, as in

$$rest \rightarrow + \;\; term \;\; \{print('+')\}\; rest_1 \tag{3.1}$$

We shall see such rules when we consider an alternative form of grammar for expressions, where the non-terminal *rest* (equation 3.1) represents "everything but the first term of an expression." Again, the subscript in $rest_1$ distinguishes this instance of non-terminal *rest* in the production body from the instance of *rest* at the head of the production.

When drawing a parse tree for a translation scheme, we indicate an action by constructing an extra child for it, connected by a dashed line to the node that corresponds to the head of the production. For example, the portion of the parse tree for the above production and action is shown in Fig. 3.1. The node for a semantic action has no children, so the action is performed when this node is first seen.

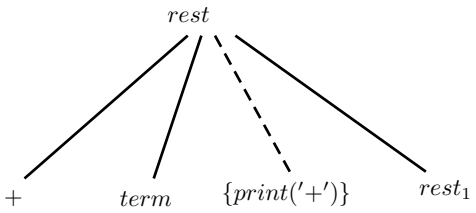

Figure 3.1: An extra leaf ($\{print('+')\}$) is constructed for a semantic action.

Definition 3.1 Preorder. *The Preorder list of a (sub)tree rooted at node N consists of: N, followed by the* preorders *of the subtrees of each of its children, if any, from the left.*

Definition 3.2 Postorder. *The postorder of a (sub)tree rooted at N consists of the postorders of each of the subtrees for the children of N, if any, from the left, then followed by N itself.*

Example 3.3 *Actions translating* $7 - 3 + 5$ *into* $7\ 3 - 5\ +$.

Solution. The parse tree in Fig. 3.2 has print statements at extra leaves, which are attached by dashed lines to interior nodes of the parse tree. The translation scheme appears in

Table 3.1. The underlying grammar generates expressions consisting of digits separated by plus and minus signs. The actions embedded in the production bodies translate such expressions into postfix notation, provided we perform a left-to-right depth-first traversal of the tree and execute each print statement when we visit its leaf.

Table 3.1: Actions for translating into postfix notation

$expr$	$\rightarrow$	$expr_1 + term$	$\{print('+')\}$
$expr$	$\rightarrow$	$expr_1 - term$	$\{print('-')\}$
$expr$	$\rightarrow$	$term$	
$term$	$\rightarrow$	0 $\{print('0')\}$	
$term$	$\rightarrow$	1 $\{print('1')\}$	
...	...	...	
$term$	$\rightarrow$	9 $\{print('9')\}$	

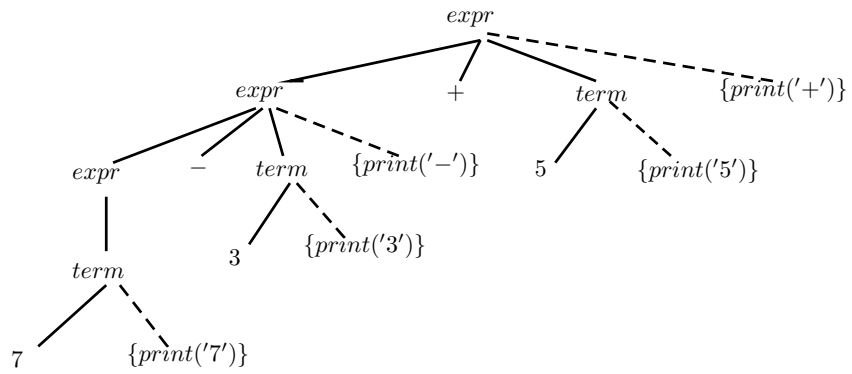

Figure 3.2: Actions translating $7 - 3 + 5$ into $7\ 3 - 5\ +$.

The root of Fig. 3.2 represents the first production in Tables. 3.1. In a postorder traversal, we first perform all the actions in the leftmost subtree of the root, for the left operand, also labeled *expr* like the root. We then visit the leaf '+' at which there is no action. We next perform the actions in the subtree for the right operand *term* and, finally, the semantic action {print('+')} at the extra node.

Since the productions for *term* have only a digit on the right side, that digit is printed by the actions for the productions. No output is necessary for the production $expr \rightarrow term$, and only the operator needs to be printed in the action for each of the first two productions. When executed during a postorder traversal of the parse tree, the actions in Fig. 3.2 print $7\ 3 - 5\ +$. □

3.2.1 Exercises

1. Construct a syntax-directed translation scheme that translates arithmetic expressions from infix notation into prefix notation in which an operator appears before its operands; e.g., $-xy$ is the prefix notation for $x - y$. Give annotated parse trees for the inputs $7 - 3 + 5$ and $7 - 3 * 5$.

Ans. Given Productions:

$$\begin{aligned} expr &\rightarrow expr + term \\ &\mid expr - term \\ &\mid term \end{aligned}$$

Translations schemes from infix notation into prefix are:

$$\begin{aligned} expr &\rightarrow \{print(\text{“}+\text{”})\}\ expr + term \\ &\mid \{print(\text{“}-\text{”})\}\ expr - term \\ &\mid term \end{aligned}$$

Ans. Given Productions:

$$\begin{aligned} expr &\rightarrow expr * term \\ &\mid expr/term \\ &\mid term \end{aligned}$$

Translations schemes from infix notation into prefix are:

$$\begin{aligned} expr &\rightarrow \{print(\text{“}*\text{”})\}\ expr * term \\ &\mid \{print(\text{“}/\text{”})\}\ expr/term \\ &\mid factor \\ factor &\rightarrow digit\ \{print(digit)\} \\ &\mid (expr) \end{aligned}$$

The complete answer is Ans_1 plus Ans_2.

2. Construct a syntax-directed translation scheme that translates arithmetic expressions from postfix notation into infix notation. Give annotated parse trees for the inputs $7\ 3 - 5\ *$ and $7\ 3\ 5\ * -$.
3. Construct a syntax-directed translation scheme that translates integers into roman numerals.
4. Construct a syntax-directed translation scheme that translates roman numerals into integers.
5. Construct a syntax-directed translation scheme that translates postfix arithmetic expressions into equivalent prefix arithmetic expressions.

Ans. Given production is:

$$expr \rightarrow expr\ expr\ op \mid digit$$

Translation scheme:

$$expr \rightarrow \{print(op)\}\ expr\ expr\ op \mid digit\ \{print(digit)\}$$

6. Explain the difference between *synthesized* and *inherited* attributed attributes. Give examples for each.
7. Define following:
   (a) Syntax Directed Definition (SDD)
   (b) Syntax Directed Translation (SDT)
   (c) Semantic Actions
8. Define preorder and postorder traversals of trees. Give one example of each. FInd out the time and space complexities of standard free and post order travels.