7CS5A COMPILER CONSTRUCTION (Syntax Directed Translation) Fall 2018

Lecture 1: Sept. 27, 2018

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

1.1 Grammars and Association Rules

We are interested in translation of languages generated by context-free grammars (CFG). This translation is useful for type checking and intermediate code generation, and also useful for implementation of little languages for specialized tasks. In the following, we get our selves familiarized by certain nomenclature/terms, which are essential for understanding of syntax directed translation.

1.1.1 Parsing

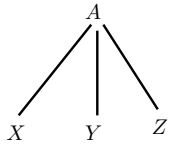

Figure 1.1: Parse-tree for $A \rightarrow XYZ$.

$A \rightarrow X_1, X_2, ..., X_n$, where X_i is a terminal or non-terminal. As a special case $A \rightarrow \varepsilon$, then A may have a single child labeled as ε.

Consider the production rules:

$$\begin{aligned} list &\rightarrow list + digit \\ list &\rightarrow list - digit \\ list &\rightarrow digit \\ digit &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9. \end{aligned}$$

Using the above rules, the expression: $7 - 3 + 5$ can be generated, for which the parse-tree is shown in Figure 1.2.

1.1.2 Association Rules

Note that $7 - 3 + 5$ is equal to $(7 - 3) + 5$ as well to $7 - (3 + 5)$, depending on how the association rules are defined. The operator + associate to left, because an operand with

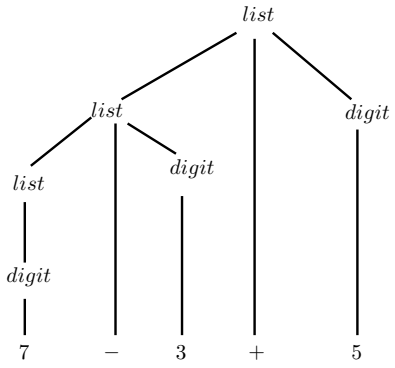

Figure 1.2: Parse-tree for $7 - 3 + 5$.

plus signs on both sides of it belongs to the operator to its left. In most programming languages the four arithmetic operators, addition, subtraction, multiplication, and division are left-associative.

Some common operators such as exponentiation are right-associative, for example, the assignment operator $a = (b = c)$. Strings like $a = b = c$ with a right-associative operator are generated by the following grammar:

$$\begin{aligned} &right \rightarrow letter = right \mid letter \\ &letter \rightarrow a \mid b \mid ... \mid z \end{aligned} \tag{1.1}$$

The difference between a parse tree for a left-associative operator like $'-'$ and a parse tree for a right-associative operator like $' ='$ is shown by Fig. 1.3. Note that the parse tree for $7 - 3 - 5$ grows down towards the left, whereas the parse tree for $a = b = c$ grows down towards the right.

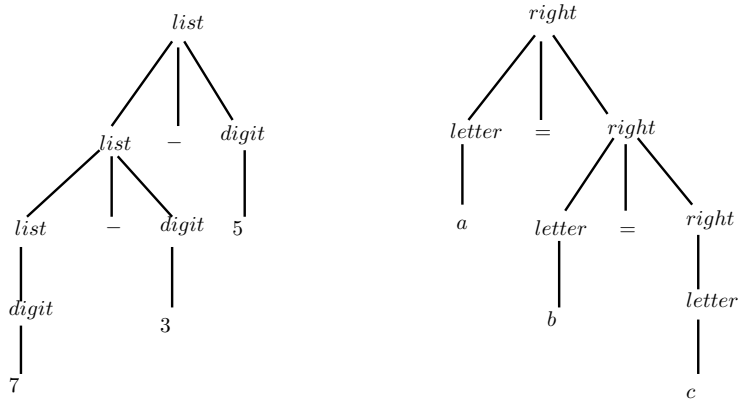

Figure 1.3: Parse-trees for left associative expression:$7-3-5$ and right-associative expression $a = b = c$.

A grammar for arithmetic expressions can be constructed from a table showing the *associativity* and *precedence* of operators. We start with the four common arithmetic operators (+, -, *, /) and a precedence Table 1.1, showing the operators in order of increasing precedence.

Table 1.1: Precedence Table.

Type of Association	Operators
Left-associative	$+, -$
Left-associative	$*, /$

In this table, operators on the same line have the same associativity and precedence:

Now consider the binary operators, $*$ and $/$, that have the higher precedence than '+' and '-'. Since these operators associate to the left, the productions are similar to those for lists that associate to the left. The nonterminal *term* is for two levels of precedence.

$$\begin{aligned} term &\rightarrow term * factor \\ &\mid term/factor \\ &\mid factor. \end{aligned} \tag{1.2}$$

Next we create the nonterminals *expr* for the two levels of precedence (equation 1.4), and an extra nonterminal $factor$ for generating basic units in expressions. The basic units in expressions are presently digits and parenthesized expressions.

$$factor \rightarrow \textbf{digit} \mid (expr) \tag{1.3}$$

Similarly, *expr* generates lists of terms separated by the additive operators.

$$\begin{aligned} expr &\rightarrow expr + term \\ &\mid expr - term \\ &\mid term. \end{aligned} \tag{1.4}$$

The definition of grammar in the form of *term*, $factor$, and *expr* enforces the precedence rules of $+, -, *$ and $/$, as well the left association.

Example 1.1 *Generate following expressions using the rules of expr, term, and* $factor$*.*

1. $3 * (5 + 2)$

2. $4/5 - 7 * 4$

Solution. We drive one expression using the production rules discussed above.

$$
\begin{aligned}
term &\Rightarrow term * factor \\
&\Rightarrow factor * factor \\
&\Rightarrow digit * factor \\
&\Rightarrow 3 * factor \\
&\Rightarrow 3 * (expr) \\
&\Rightarrow 3 * (expr + term) \\
&\Rightarrow 3 * (term + term) \\
&\Rightarrow 3 * (factor + term) \\
&\Rightarrow 3 * (digit + term) \\
&\Rightarrow 3 * (5 + term) \\
&\Rightarrow 3 * (5 + digit) \\
&\Rightarrow 3 * (5 + 2)
\end{aligned}
$$

1.1.3 Term, expression and factor

The difference between the term, expression and factor is as follows: A factor is an expression that cannot be torn apart by any operator. If a factor is a parenthesized expression, the parenthesis protect against such tearing.

A term that is not factor can be torn apart by operators of highest precedence: $*$ and $/$, and not by the lower precedence operators $(+, -)$. And, an expression that is not a term or factor can be torn apart any any operator out of four.

Example 1.2 *Production rule for java statement.*

The keywords allow us to recognize statements, since most statements begin with keyword or special character. Exception to this are assignment and procedure call statements. The below given is a production rule for java statement.

$$
\begin{aligned}
stmt \rightarrow\ & \textbf{id} = expression; \\
& \mid \textbf{if}\ (expression)\ stmt \\
& \mid \textbf{if}\ (expression)\ stmt\ \textbf{else}\ stmt \\
& \mid \textbf{while}\ (expression)\ stmt \\
& \mid \textbf{do}\ stmt\ \textbf{while}\ (expression); \\
& \{stmt\}
\end{aligned}
\tag{1.5}
$$

$$
\begin{aligned}
stmt \rightarrow\ & stmt\ \ stmt \\
& \mid \varepsilon
\end{aligned}
\tag{1.6}
$$

□

1.1.4 Exercises

1. What are the left associative and right associative operators? Give two other examples for each, apart from what has been discussed in this lecture note.
2. Define *term*, *expression*, and $factor$, using their built-in recursive structures. Construct any two expressions making use of all these three.
3. Construct parse trees for following using definitions of *term*, *expr*, and *factor*.
   (a) $(3 + 5 + 2)/(7 - 3 + 5)$
   (b) $(3 + 5/(2 + 5))$
   (c) $((7 + 5) * (5 + 7)/5)$
4. Answer each in brief.
   (a) What do you mean by tearing-off the expression?
   (b) What is fundamental difference between *term* and *factor*.
   (c) Justify the statement: "An expression that is not a term or factor, can be torn apart by any operator." Give examples.
   (d) Give examples of statements that do not begin with keywords.
5. Consider the CFG: $S \rightarrow S\ S\ + \mid S\ S\ * \mid a$.
   (a) Show generating the string $aa + a*$ using this grammar.
   (b) Construct parse-tree for $aa + a*$.
   (c) Describe the language generated by this grammar.
6. What languages are generated by the following grammars?
   (a) $S \rightarrow 0\ S\ 1 \mid 01$
   (b) $S\ (\ S\ )\ S \mid \varepsilon$
   (c) $S \rightarrow +\ S\ S \mid -\ S\ S \mid a$
7. Prove that that all binary strings generated by the following grammar have values divisible by 3.parse tree.

$$num \rightarrow 11 \mid 1001 \mid num\ 0 \mid num\ num$$

8. What are the applications of syntax directed translation scheme?