Indian Institute of Information Technology
Design and Manufacturing, Kancheepuram
Chennai 600 127, India
An Autonomous Institute under MHRD, Govt of India
http://www.iiitdm.ac.in
COM306T-Automata and Compiler Design

Instructor
Dr. V. Masilamani
Scribe: Vignesh Sairaj

Lecture-36 `Syntax tree construction from annotated parse tree using post order traversal, and also during parsing(using SDD)`

1 Introduction

1.1 Parse Tree (Concrete Syntax Tree)

Given a CFG, a Parse tree is any ordered tree that satisfies the following properties:

- The root is labeled by the start symbol.
- Each leaf is labeled by a terminal or by ϵ.
- Each interior node is labeled by a nonterminal.
- If A is the nonterminal labeling some interior node and $X_1, X_2, ..., X_n$ are the labels of the children of that node from left to right, then there must be a production $A \rightarrow X_1X_2...X_n$.

Consider the following CFG G:

$$list \rightarrow list + digit$$
$$list \rightarrow list - digit$$
$$list \rightarrow digit$$
$$digit \rightarrow 0|1|2|3|4|5|6|7|8|9$$

Figure 1 is a parse tree for the above grammar. The expression $9 - 9 + 2$ is called the yield of the parse tree. It is the string formed by concatenating the labels of the leaf nodes of the tree.

1.1.1 Parsing

The parse tree is pictorial representation of the derivation of the yield string from the grammar. Parsing is the process of constructing the parse tree (explicitly or implicitly) given the CFG and the input string. An unambigous grammar is one for which there are no two distinct parse trees that yield a given string. Such grammars are useful as the parse tree can actually be used to attach meaning to the programming constructs (and the symbols that represent the constructs). It is imperative that there is only one meaning assignable to a given legal expression.

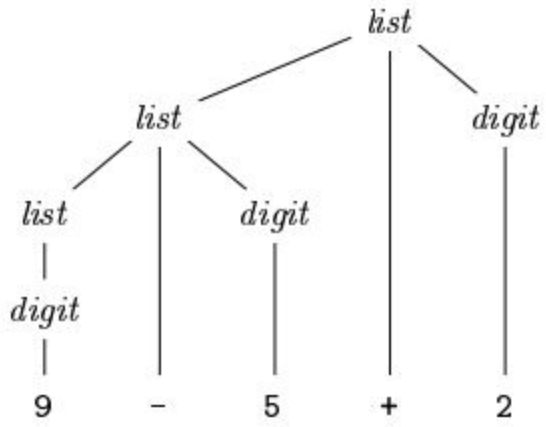

Figure 1: Parse Tree for G

2 Syntax-Directed Definition

2.1 Attributes

An attribute is any quantity associated with a programming construct (or grammar symbols (nonterminals and terminals) used to represent programming constructs). Examples of attributes are data types of expressions, the number of instructions in the generated code, or the location of the first instruction in the generated code for a construct, among many other possibilities.

2.2 Syntax-Directed Translation Schemes

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

2.3 Syntax Directed Definition

A syntax-directed definition associates

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

e.g.

PRODUCTION	*SEMANTICRULES*
$expr \rightarrow expr_1 + term$	$expr.t = expr_1.t \| \| term.t \| \| '+'$
$expr \rightarrow expr_1 - term$	$expr.t = expr_1.t \| \| term.t \| \| '-'$
$expr \rightarrow term$	$expr.t = term.t$
$term \rightarrow 0$	$term.t = '0'$
$term \rightarrow 1$	$term.t = '0'$
...	...
$term \rightarrow 9$	$term.t = '9'$

Table 1: Syntax-directed definition for infix to postfix translation

Table 1 shows a syntax-directed definition for infix to postfix translation. The attributes are evaluated either while parsing itself, or can be evaluated once the parse tree has been obtained by post-order traversal as illustrated below.

Algorithm 1 Depth First Traversal of Tree

```
function VISIT(node N)
    for each child C of N, from left to right do
        VISIT(C);
        evaluate semantic rules at node N;
    end for
end function
```

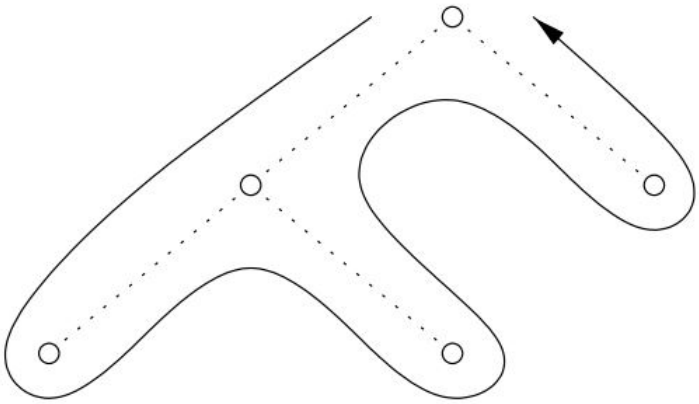

Figure 2: Example of a depth-First traversal of a tree

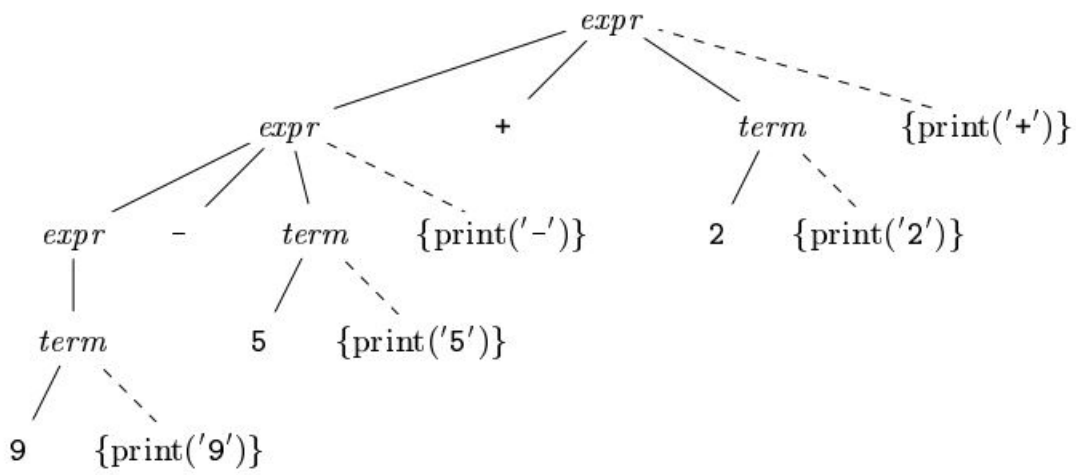

Figure 3: Actions translating $9-5+2$ into $95-2+$

3 Intermediate Code Generation

Once a parse tree has been generated, it is passed to a *Semantic Analyzer* which annotates the parse tree (evaluates it's attributes). For *synthesized* attributes, the values are evaluated in a bottom-up order, while *inherited attributes* attributes are evaluated in a top-down order.

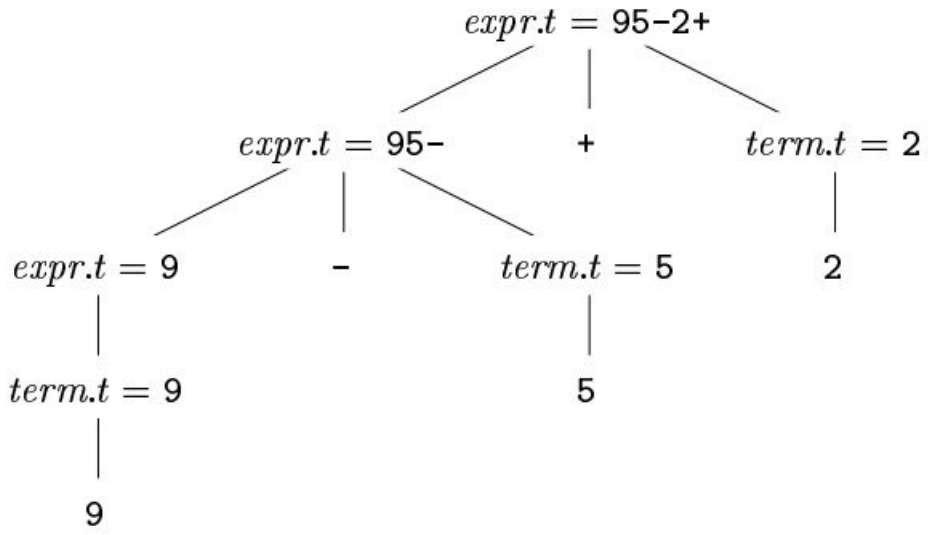

Figure 4: Attribute values at nodes in a parse tree for SDD in table 1

There are many ways of generating intermediate code, one of which is constructing a(n) *(Absract) Syntax Tree.*

4 Abstract Syntax Tree (AST)

An abstract syntax tree (AST), or just *syntax tree*, is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code. The syntax is abstract in not representing every detail appearing in the real syntax. While a Parse Tree explicitly shows how the string is derived, the Syntax Tree does away with language-specific details and retains only information relevant to further generation of code in the Target Language. The tree can be constructed by evaluating nodes/labels as attributes of the grammar symbols during parsing (using an SDD) or from the parse tree by post order traversal as shown above. Here we see an example SDD (grammar H and ruleset R) that define the language (set) of legal expression strings and describe a procedure to find the nodes (treated here as attributes) of the corresponding AST.

The SDD for grammar H:

$PRODUCTION\ of\ H$	$SEMANTICRULES(R)$
$E \rightarrow E_1 + T$	$E.node = \mathbf{new}NODE(E_1.node, T.node, '+')$
$E \rightarrow E_1 - T$	$E.node = \mathbf{new}NODE(E_1.node, T.node, '-')$
$E \rightarrow T$	$E.node = T.node$
$T \rightarrow (E)$	$T.node = E.node$
$T \rightarrow \mathbf{id}$	$T.node = \mathbf{new}LEAF(\mathbf{id}, \mathbf{id}.entry$
$T \rightarrow \mathbf{num}$	$T.node = \mathbf{new}LEAF(\mathbf{num}, \mathbf{num}.entry$

Table 2: Constructing syntax trees for simple expressions using and SDD

5 References

All examples and illustrations were sourced from: Aho, Alfred V.; Lam, Monica S.; Sethi, Ravi; Ullman, Jeffrey D. ***Compilers, Principles, Techniques, and Tools (Second edition).***

6 Gate Questions

6.1 Gate

In the context of abstract-syntax-tree (AST) and control-flow-graph (CFG), which one of the following is True?

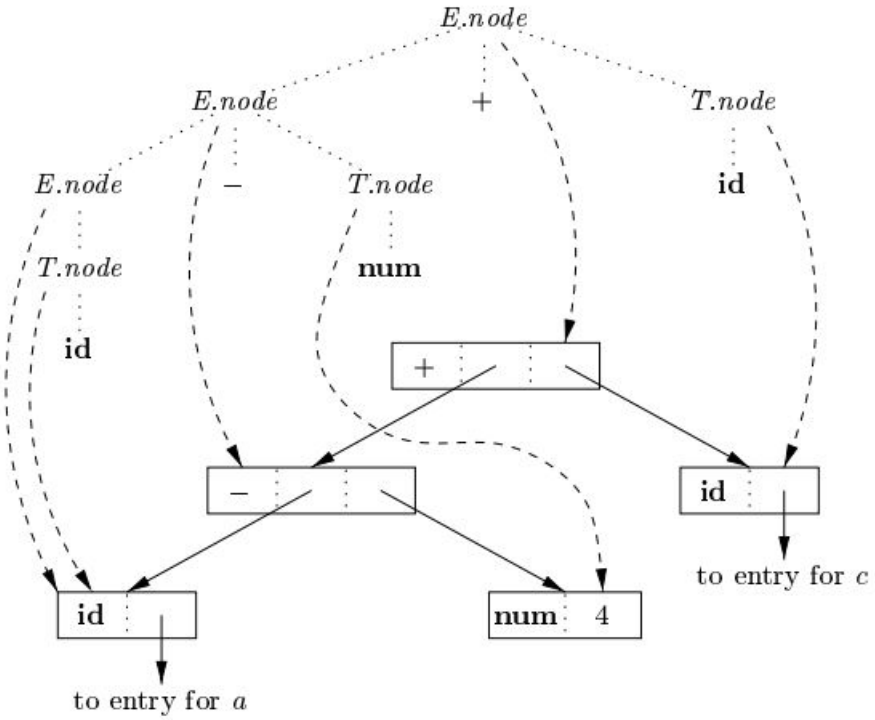

Figure 5: Syntax tree for $a - 4 + c$ using SDD in Table 2

A In both AST and CFG, let node N2 be the successor of node N1. In the input program, the code corresponding to N2 is present after the code corresponding to N1

B For any input program, neither AST nor CFG will contain a cycle

C The maximum number of successors of a node in an AST and a CFG depends on the input program

D Each node in AST and CFG corresponds to at most one statement in the input program

The answer is **(C)**, as:

A control flow graph (CFG) is a representation, using graph notation, of all paths that might be traversed through a program during its execution. Each node corresponds to a block of code, and each edge corresponds to the transfer of control between blocks of code.

A is false, In a CFG (Control Flow Graph), the code of N2 may be present before N1 when there is a loop or goto.

B is false, a CFG (Control Flow Graph) contains cycle when input program has loop.

C is true, successors of ASTs and CFGs depend on input program

D is false, a single node may correspond to a block of statements.