Report

Topic: Syntax Directed Definitions and Syntax Directed Translation (28-oct)

- KUMAR SAURAV (11CS30016)

Syntax-Directed Definitions

A syntax-directed definition (SDD) is a context-free grammar together with, attributes and rules.

Inherited and Synthesized Attributes:

1) A synthesized attribute for a nonterminal A at a parse-tree node N is defined by a semantic rule associated with the production at N. A synthesized attribute at node N is defined only in terms of attribute values at the children of N and at N itself.

2) An inherited attribute for a nonterminal B at a parse-tree node N is defined by a semantic rule associated with the production at the parent of N. An inherited attribute at node N is defined only in terms of attribute values at N's parent, N itself, and N's siblings.

Application of SDD:

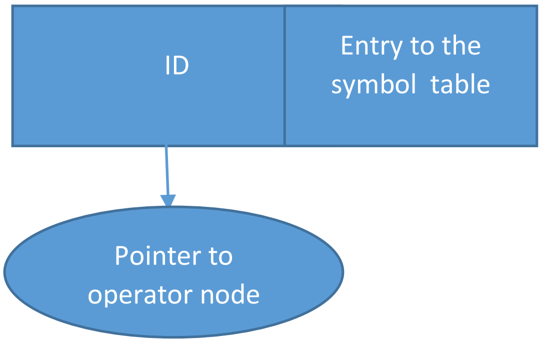
a. Construction of data structure : syntax tree

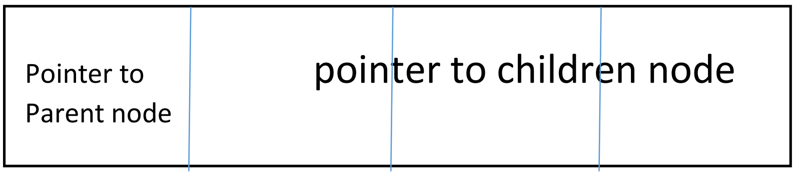
Implementation of SDD: Constructing syntax tree simultaneous with parse tree.

Operand:

i) Identifier (entry is the pointer to the entry in the symbol table)

ii) Literal (entry = value returned from lexical analyzer)





LEAF (op, val) : Creates a leaf node for the syntax tree.

NODE (op, c_1, c_2, c_3….., c_k) : Creates an internal node having parent operator op and children c_1, c_2, c_3….., c_k.

Examples

Production Rules:

- E -----> E_1 + T
- E -----> E – T
- E -----> T
- T -----> id
- T -----> num

For 4) & 5), create Leaf node, because id & num are terminals
For 1) ,2) , create internal node

Semantics Rules:

- E -----> E_1 + T [E.node_addr = new Node(+,E_1.node_addr,T.node_addr)]
- E -----> E – T [E.node_addr = new Node(-, E.node_addr,T.node_addr)]
- E -----> T [E.node_addr = new T.node_addr]
- T -----> id [T.node_addr = new Leaf(id, entry in symbol table)]
- T ------> num [T.node_addr = new Leaf(num, val)]

Evaluation of SDD

1) Construct Parse Tree
2) Post order Transversal

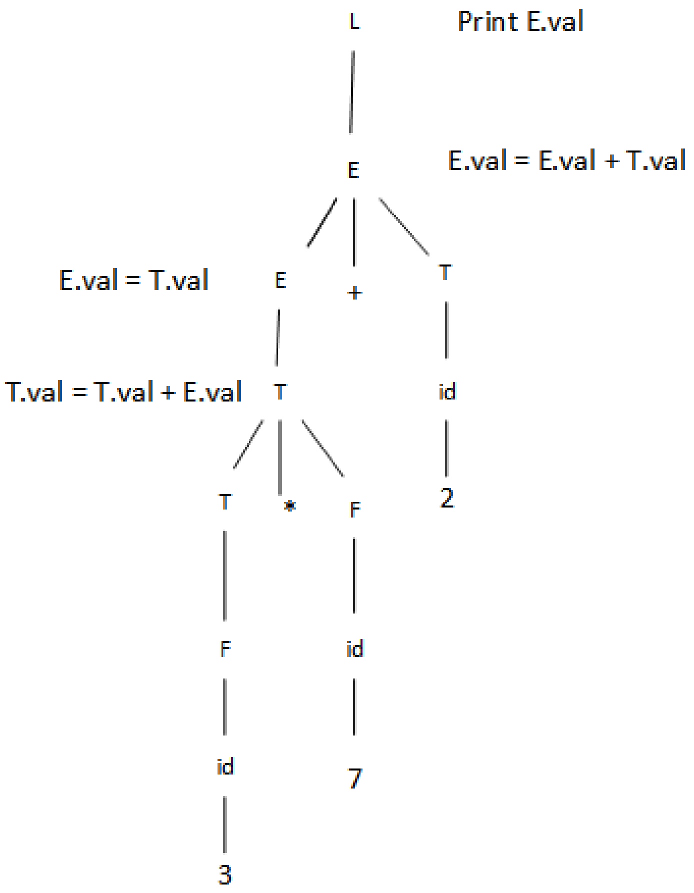
SDT => Embed print part of code within production
Eg., A ---> xy{a}Z ; For SDT we introduce Print in each rule.

Example:

- L ---> E {Print (E.val) }
- E ---> E_1 + T {E.val = E_1.val + T.val }
- E ---> T {E.val = T.val}
- T ---> T_1 * F {T_1 .val = T.val * F.val }
- T ---> F {T.val = F.val}
- F ---> id {F.val = id}

Evaluate: 3*7+2

1) Construct Parse tree dropping the code fragment
2) Add code fragment maintaining the order
3) Post order Transversal



S-attributed SDD : Semantic Rules => Compute synthesized attributes from children

1) Translate these rules to equivalent code fragment
2) Place those action at the end of production