**Syntax Directed Translation Schemes**

SDT is a complementary notation to SDD. All applications of SDD can be implemented using SDT. SDT is a context-free grammar with program fragments called semantic actions embedded within production bodies.

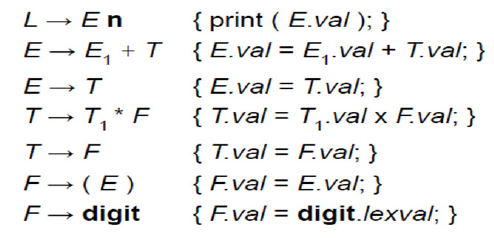
Any SDT can be implemented by first building a parse tree and then performing the actions in a left-to-right depth-first order i.e. during a pre-order traversal. Typically SDT's are implemented during parsing without building parse tree. During parsing, an action in a production body is executed as soon as all the grammar symbols to the left of action have been matched.

**Postfix Translation Schemes**

The simplest implementation of SDD occurs when we can parse the grammar bottom-up and SDD is S-attributed. Here, each semantic action can be placed at the end of production and executed along with the reduction of body to the head of the production.

This type of SDT is called *Postfix SDT.*

Example: Postfix SDT for implementing the desk calculator is as below.



**Parser-Stack Implementation of Postfix SDTs**

Postfix SDT’s can be implemented during LR parsing. Attribute of each grammar

symbol can be put on the stack in place where they can be found during reduction, i.e.. place attribute along with grammar symbol in record of stack itself.

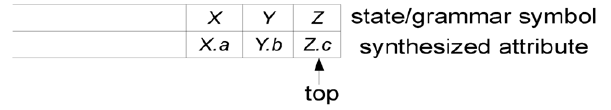


Fig. parser stack with field for synthesised attributes

To implement SDT during LR parsing, add semantic stack parallel to the parsing stack: each symbol (terminal or non-terminal) on the parsing stack stores its value on the semantic stack. It holds terminals’ attributes and nonterminals’ translations. When the parse is finished, the semantic stack will hold just one value: the translation of the root non-terminal (which is the translation of the whole input).

**Semantic Actions during Parsing**

• when shifting

– push the value of the terminal on the semantic stack

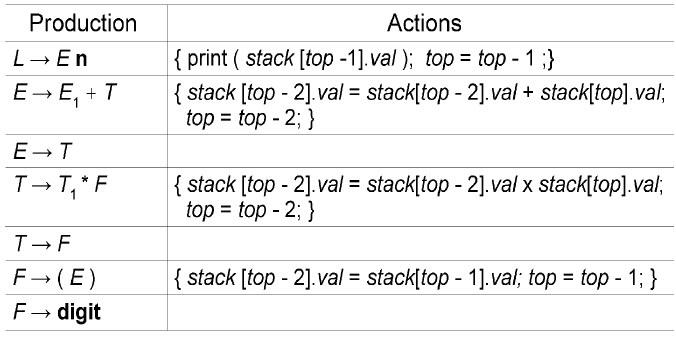
• when reducing

– pop k values from the semantic stack, where k is the number of symbols

on production’s RHS

– push the production’s value on the semantic stack

The SDT for implementing the desk calculator on a bottom-up parsing stack is as above.



**SDTs with Actions inside Productions**

• Action can be placed at any position in the production body.

• Action is performed immediately after all symbols left to it are processed.

• Given B —> X { a } Y , an action a is done after

– we have recognized X (if X is a terminal), or

– all terminals derived from X (if X is a nonterminal).

• If bottom-up parser is used, then action a is performed as soon as X appears on

top of the stack.

• If top-down parser is used, then action a is performed

– just before Y is expanded (if Y is nonterminal), or

– check Y on input (if Y is a terminal).

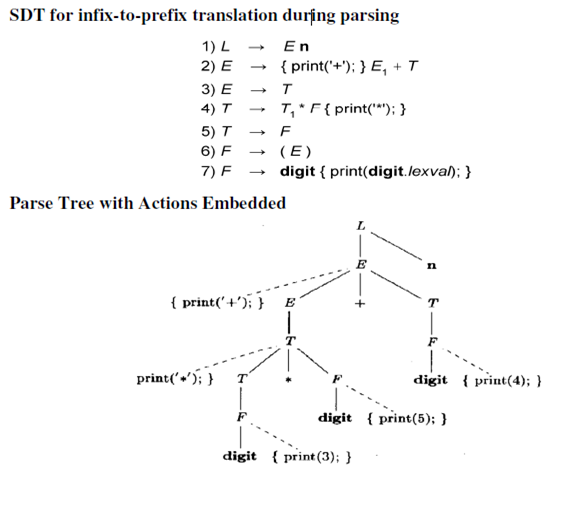
• Any SDT can be implemented as follows:

– Ignoring actions, parse input and produce parse tree.

– Add additional children to node N for action in \_, where A \_ \_.

– Perform preorder traversal of the tree, and as soon as a node labeled by an

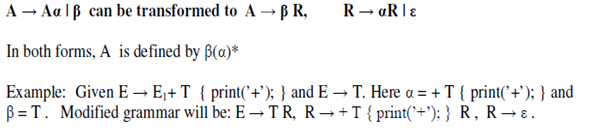
action is visited, perform that action.



**Eliminating Left Recursion from SDTs**

Grammar with left recursion cannot be parsed using top-down parser. In case of SDT we treat action as terminal symbol in the production. Then we use the following rule of

transforming grammar to non left-recursive form.



Example :

Rewrite the following SDT so that the underlying grammar becomes non-left-recursive.

A → A{a}B │AB{b}│0

B → B{c}A │BA{d}│1

Solution :

A→0A1

A1→{a}BA1│B{b}A1│ ε

B→1B1

B1→{c}A B1│A{d} B1│ ε

**SDTs for L-Attributed Definitions**

• It is necessary that the underlying grammar can be parsed top-down

• Rules to modify L-attributed SDD to SDT

– Embed the action for computing inherited attributes for a nonterminal A

immediately before A in production body

– Place the action for computing synthesized attribute for the head at the end

of the body of production

**Example :** This example is about the generation of intermediate code for a

typical programming-language construct: a form of while-statement.

