Semantic Rules with side effects

Example

Let's consider the following grammar:

$D \rightarrow TL; \quad T \rightarrow \textbf{int}|\textbf{float} \quad L \rightarrow L_1, \textbf{id}|\textbf{id}$

Let's add sematic rules to successively permit type checking

Semantic Rules with side effects

Exercise

Let's consider the following grammar that generates binary numbers with a decimal point:

$S \to L.L|L \quad L \to LB|B \quad B \to 0|1$

Design an L-attributed and an S-attributed SDD to make the translation in decimal numbers

Abstract Syntax Tree

Abstract Syntax Tree

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. For instance, grouping parentheses are implicit in the tree structure, and a syntactic construct like an if-condition-then expression may be denoted by means of a single node with three branches.
Syntax trees are useful for translation purpose making the phase much easier.

Let's consider the sentence $(a + b) * 5$ over the grammar:

$E \rightarrow TE' \quad E' \rightarrow +TE'|\epsilon \quad T \rightarrow FT' \quad T' \rightarrow *FT'|\epsilon \quad F \rightarrow (E)|\mathbf{id}|\mathbf{num}$

Let's build the parse tree and the AST

Using SDT to build AST

To build a syntax tree two different kind of nodes need to be created, the leaves ($Leaf(op, val)$) and the internal nodes ($Node(op, c_1, \ldots, c_n)$). In the following consider the sentence $a - 4 + c$.

1. Let's built an SDD with actions permitting to derive the syntax tree for expressions grammar in the form suitable for LR parsing.
   $E \to E_1 + T$, $E \to E_1 - T$, $E \to T$, $T \to (E)$, $T \to$ **id**, $T \to$ **num**
2. Let's repeat the exercise for an expression grammar parsable by LL parsers.
   $E \to TE'$, $E' \to +TE'_1$, $E' \to -TE'_1$, $E' \to \epsilon$, $T \to (E)$, $T \to$ **id**, $T \to$ **num**

Towards type checking

Let's now consider the case of a grammar for type definition:
$T \to BC$, $B \to$ **int**, $B \to$ **float**, $C \to$ [**num**]C, $C \to \epsilon$
Define sematics rules to assign a type to an expression and try it on the sentence:

int[2][3]