SDD with acyclic topological sort

S-attributed

If every attribute is synthesized the SDD is said S-attributed, in such a case an LR parser could even avoid the explicit derivation of the parse tree

L-attributed

Each attribute in the SDD satisfies one of the following conditions:

- it is synthesized
- it is inherited but it depends only from attributes on siblings on the left or inherited attributes associated to the parent symbol
- it is inherited or synthesized from attributes from the same symbol in a way that cycles are not generated

Semantic rules with controlled side effects

Side effects

A side effect consists of program fragment contained with semantic rules. It is necessary to control side effects is SDD in two possible ways:

- Permit incidental side effects
- Constraint admissible evaluation orders so to have the same translation with any admissible order.

Why to use them?

- to associate actions to carry on with specific steps of the compiler
- to print messages for the user useful during compilation
- to check correctness related aspects (e.g. types)

Semantic rules with controlled side effects

Side effects

A side effect consists of program fragment contained with semantic rules. It is necessary to control side effects is SDD in two possible ways:

- Permit incidental side effects
- Constraint admissible evaluation orders so to have the same translation with any admissible order.

Why to use them?

- to associate actions to carry on with specific steps of the compiler
- to print messages for the user useful during compilation
- to check correctness related aspects (e.g. types)

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:

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

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:

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

Let's build the parse tree and the AST

Using SDDs 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 ($Node(op, c_1, \dots, c_n)$). 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.
   $E \rightarrow E_1 + T$, $E \rightarrow E_1 - T$, $E \rightarrow T$, $T \rightarrow (E)$, $T \rightarrow \mathbf{id}$, $T \rightarrow \mathbf{num}$
2. Let's repeat the exercise for an expression grammar parsable by LL parsers.
   $E \rightarrow TE'$, $E' \rightarrow +TE'_1$, $E' \rightarrow -TE'_1$, $E' \rightarrow \epsilon$, $T \rightarrow (E)$, $T \rightarrow \mathbf{id}$, $T \rightarrow \mathbf{num}$

Towards type checking

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

$$\mathbf{int}[2][3]$$