Semantic analysis

Semantic analysis

- Semantic analysis checks for the correctness of contextual dependences:
  - finds correspondence between declarations and usage of identifiers,
  - performs type checking/inference,
  - ...
- Syntax tree is decorated with typing- and other context dependent information.

Semantic analysis

- Semantic analysis checks restrictions imposed by a static semantics of the language.
- Sometimes it is possible to expess semantic properties by context-free grammars, but usually this puts heavy restrictions to the language and/or complicates the grammar.
- Example – simple typed expressions:

$$
\begin{array}{lcl}
\text{IntExp} & \rightarrow & int \mid intVar \\
 & \mid & \text{IntExp} + \text{IntExp} \\
\text{BoolExp} & \rightarrow & true \mid false \\
 & \mid & boolVar \\
 & \mid & \text{IntExp} \leq \text{IntExp} \\
 & \mid & not\ \text{BoolExp} \\
 & \mid & \text{BoolExp}\ \&\ \text{BoolExp}
\end{array}
$$

Semantic analysis

- At first glance, the grammar looks reasonable, but:
  - the grammar has two different (lexical) classes for variables;
  - additional types require new classes of variables;
  - most languages do not put restrictions to variable names based on their types;
  - moreover, usually one is allowed to use the same variable name for variables of different types in different context.

Attribute grammars

- Attribute grammars are generalization of context-free grammars, where:
  - each grammar symbol has an associated set of attributes;
  - each production rule has a set of attribute evaluation rules (or semantic rules).
- The goal is to find an evaluation of attributes which is consistent with the given semantic rules.

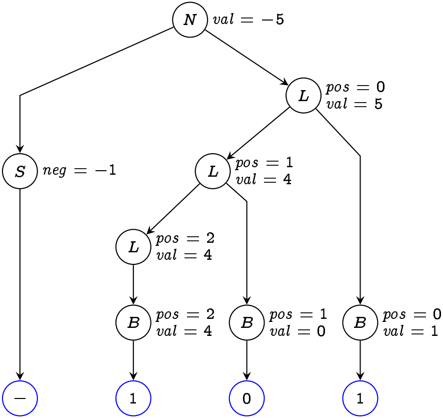
Attribute grammars

Example:

Productions	Semantic rules
$N \rightarrow S\ L$	$L.pos := 0$ $N.val := S.neg * L.val$
$S \rightarrow +$	$S.neg := 1$
$S \rightarrow -$	$S.neg := -1$
$L \rightarrow L_1\ B$	$L_1.pos := L.pos + 1$ $B.pos := L.pos$ $L.val := L_1.val + B.val$
$L \rightarrow B$	$B.pos := L.pos$ $L.val := B.val$
$B \rightarrow 0$ $B \rightarrow 1$	$B.val := 0$ $B.val := 2^{B.pos}$

Attribute grammars

Example:



Attribute grammars

- Semantic rules associated with a production $A \to \alpha$ are in the form $y = f(x_1, \ldots, x_n)$, where y and x_i are attributes associated with symbols in the production, and f is a function.

- There are two kinds of attributes:
  - synthesized attributes: y is an attribute associated with the non-terminal A;
  - inherited attributes: y is an attribute associated with some symbol in α.

- Synthesized attributes depend only attribute values of the subtrees.

- Inherited attributes may depend from the values of parent node and siblings.

Attribute grammars

Example:

Productions	Semantic rules
$N \rightarrow S\ L$	$L.pos := 0$
	$N.val := S.neg * L.val$
$S \rightarrow +$	$S.neg := 1$
$S \rightarrow -$	$S.neg := -1$
$L \rightarrow L_1\ B$	$L_1.pos := L.pos + 1$
	$B.pos := L.pos$
	$L.val := L_1.val + B.val$
$L \rightarrow B$	$B.pos := L.pos$
	$L.val := B.val$
$B \rightarrow 0$	$B.val := 0$
$B \rightarrow 1$	$B.val := 2^{B.pos}$

synthesized attributes

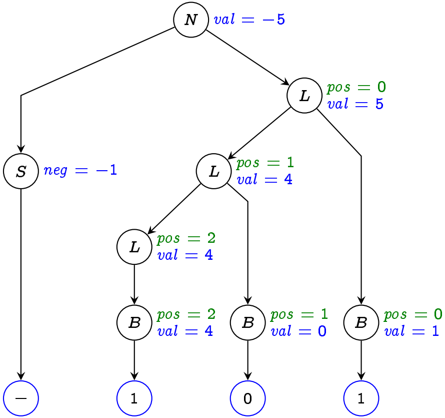
inherited attributes

Attribute grammars

- An attribute a depends from b if the evaluation of a requires the value of b.
- Dependencies between attributes define a dependency graph:
  - an directed graph, where edges show the dependencies between attributes;
  - describes the data flow during the attribute evaluation.
- Synthesized attributes have edges pointing upwards.
- Inherited attributes have edges pointing downwards and/or sidewise.

Attribute grammars

Example:



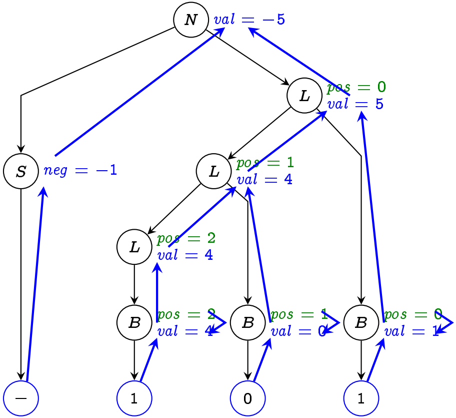
dependency graph

synthesized attributes

inherited attributes

Attribute grammars

Example:



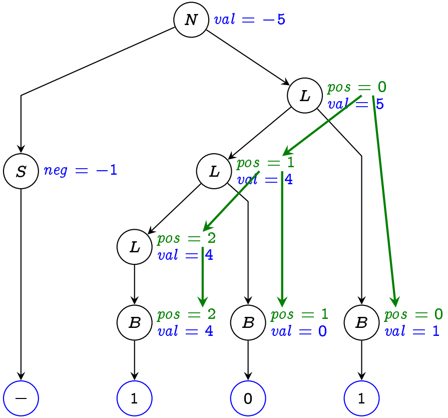
dependency graph

synthesized attributes

inherited attributes

Attribute grammars

Example:



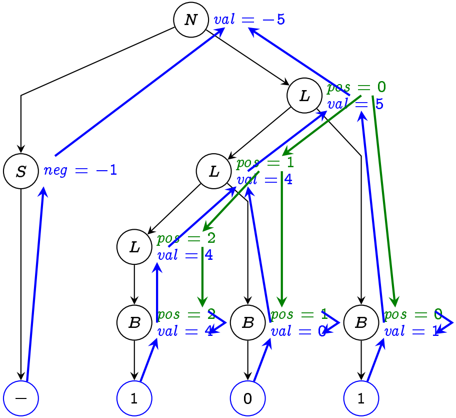
dependency graph

synthesized attributes

inherited attributes

Attribute grammars

Example:



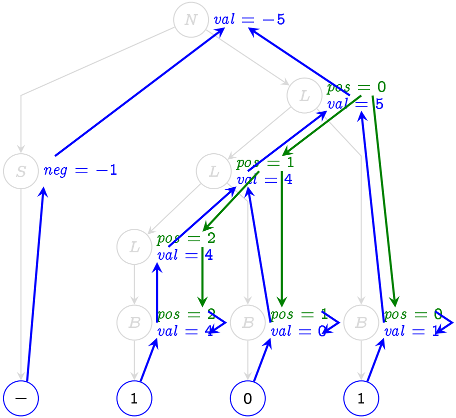
dependency graph

synthesized attributes

inherited attributes

Attribute grammars

Example:



dependency graph

synthesized attributes

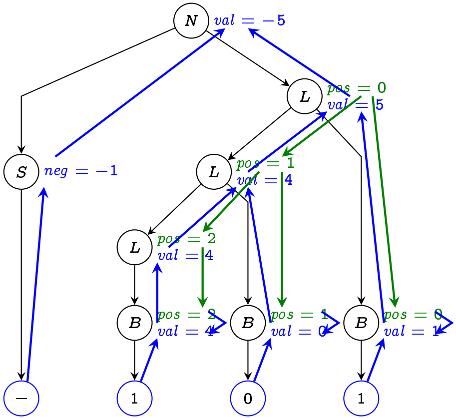
inherited attributes

Attribute grammars

- Topological sorting of a directed acyclic graph is a process of findig a linear ordering of its nodes, st., each node comes before all nodes to which it has outbound edges.
- Topological sorting of the dependency graph gives a valid evaluation ordering for attributes.
- NB! In the case of cyclic dependency graphs a valid ordering may not exist.

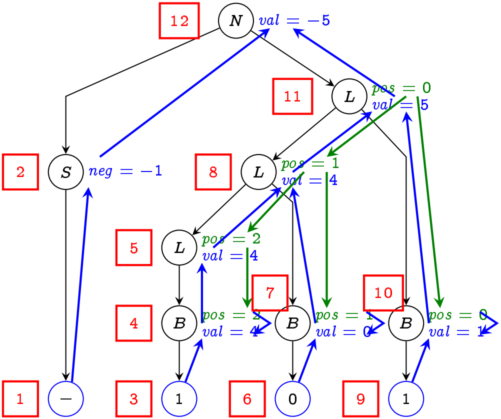
Attribute grammars

Example:



Attribute grammars

Example:



Attribute grammars

- S-attribute grammar is an AG where all attributes are synthesized.
- S-attribute grammars interact well with LR(k)-parsers since the evaluation of attributes is bottom-up.
- The values of attributes can be kept together with the associated symbol in the stack.
- Before reduction by production $A \to \alpha$, attributes corresponding to symbols of α are available in top of the stack.
- Hence, all the information for evaluating synthesized attributes of A are available, and these can be computed during reduction.

Attribute grammars

- L-attribute grammar is an AG where for all productions $A \to X_1 X_2 \ldots X_n$ inherited attributes of symbol X_i $(1 \leq i \leq n)$ depend only from inherited attributes of A and from attributes of symbols X_j $(j < i)$.
- NB! Each S-attribute grammar is also a L-attribute grammar.
- L-attribute grammars support the evaluation of attributes in depth-first left-to-right order.
- Interacts well with LL(k) parsers (both table driven and recursive decent).