Static semantics

- compile-time analysis
- syntax-directed
Nonnegative integers

<number> ::= <digit> | <number> <digit>

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Expressions

\[ <S> ::= <E> \]
\[ <E> ::= <E> + <T> \]
\[ <E> ::= <E> - <T> \]
\[ <E> ::= <T> \]
\[ <T> ::= <number> \]

What does 5 - 3 - 1 “mean”? 
Attribute grammars

Attributes: quantities (values) associated with a construct.

\( X.a \)

\( X \) is a terminal or a nonterminal

\( a \) an attribute of \( X \)

Attributes for terminal symbols come with the symbol

Attributes for nonterminal symbols are defined by semantic rules attached to productions in a grammar
A simple example

<number> ::= <digit>

<number>.val = <digit>.val

<number> ::= <number2> <digit>

<number>.val = <digit>.val
+ 10 * (<number2>.val)

<digit> ::= 0

<digit>.val = 0
Information flows bottom-up in the parse tree

Synthesized attributes
Type declarations

\[ \langle \text{vardec} \rangle \ ::= \text{int} \ \langle \text{intvlist} \rangle \]
\[ \langle \text{intvlist} \rangle \ ::= \langle \text{name} \rangle \ | \ \langle \text{name} \rangle, \ \langle \text{intvlist} \rangle \]
Inherited attributes
Semantic rules

\[
\text{vardec} ::= \text{int} \text{ intvlist} \\
\text{intvlist}.type = \text{int}.type = \text{int} \\
\text{intvlist} ::= \text{name} \\
\text{name}.type = \text{intvlist}.type \\
\text{intvlist} ::= \text{name}, \text{intvlist2} \\
\text{name}.type = \text{intvlist}.type \\
\text{intvlist2}.type = \text{intvlist}.type
\]
Attribute grammars

Every nonterminal $A$ has two sets of attributes:
- synthesized attributes $S(A)$
- inherited attributes $I(A)$

$$\text{Att}(A) = S(A) \cup I(A)$$

Every production in the grammar has associated semantic rules.

A semantic rule is of the form

*an attribute = an expression in terms of attributes*
L-attributed grammars

- Arbitrary semantic rules may cause circular dependencies
- Introduced by Lewis, Rosenkrantz & Stearns (1973)
- Attributes can be evaluated in depth-first order
  - Suitable for syntax-directed translation
L-attributed grammars

\[ A ::= A_1 \ldots A_j \ldots A_m \]

The semantic rules are of the form

\[ lhs = \text{expression} \]

(a) \( lhs \) is in \( S(A) \) or one of \( I(A_j) \)

(b) if the \( lhs \) is in \( S(A) \), then the expression is in terms of

- inherited attributes of \( A \), namely \( I(A) \), and
- all attributes of the symbols on the right-hand side of the rule

if the \( lhs \) is in \( I(A_j) \), then the expression is in terms of

- inherited attributes of \( A \), namely \( I(A) \), and
- all attributes of \( A_k \), for all \( k < j \)
invoke $A_1(\ldots)$

.. returns synthesized attributes ..
\( I_1 \) depends only on \( I \)

\( I_i \) depends on all attributes to the left of node \( A_i \)
Normalized L-attributed grammars

\[ A ::= A_1 \ldots A_j \ldots A_m \]

The semantic rules are of the form

\[ lhs = expression \]

(a) \( lhs \) is in \( S(A) \) or one of \( I(A_j) \)

(b) if the \( lhs \) is in \( S(A) \), then the expression is in terms of

- inherited attributes of \( A \), namely \( I(A) \), and

- \( synthesized \) attributes of the symbols on the right-hand side of the rule

if the \( lhs \) is in \( I(A_j) \), then the expression is in terms of

- inherited attributes of \( A \), namely \( I(A) \), and

- \( synthesized \) attributes of \( A_k \), for all \( k < j \)
Decorated parse tree

A parse tree with all attributes evaluated

- Attribute values of different *instances* of the same nonterminal may be different
Example: Conditional expressions

\[
\begin{align*}
<E> & \ ::= \ if \ <C> \ then \ <E> \ else \ <E> \\
<E>.val & \ ::= \ if \ <C>.val \ then \ <E1>.val \\
& \hspace{1em} \ else \ <E2>.val
\end{align*}
\]
Conditional Attribute Grammars

There is a boolean attribute to represent validity

Can generate non-context-free languages

\[ <S> ::= <A><B><C> \]

\[ <S>.wf = (<A>.val == <B>.val == <C>.val) \]

\[ <A> ::= a<A> \]

\[ <A>.val = <A1>.val + 1 \]

\[ <A> ::= <empty> \]

\[ <A>.val = 0 \]

Similarly for \(<B> \) and \(<C>\)
\[ <S> ::= <A><B><C> \]
\[ <S>.wf = ( <A>.val == <B>.val == <C>.val) \]
\[ <A> ::= a<A> \]
\[ <A>.val = <A1>.val + 1 \]
\[ <A> ::= <empty> \]
\[ <A>.val = 0 \]
\[ <B> ::= b<B> \]
\[ <B>.val = <B1>.val + 1 \]
\[ <B> ::= <empty> \]
\[ <B>.val = 0 \]
\[ <C> ::= c<C> \]
\[ <C>.val = <C1>.val + 1 \]
\[ <C> ::= <empty> \]