Static Semantics

Principles of Programming Languages

Fatma Başak Aydemir

basak.aydemir@boun.edu.tr

Recap

BNF

What does a program in a specific language look like?

There are some properties difficult or impossible to express with BNF.

Type compatibility rules

In an assignment statement, LHS and RHS must have the same type.
Difficult, requires duplication of rules.

$$
\begin{aligned}
\langle\text{assign-int}\rangle &\rightarrow \langle\text{var-int}\rangle = \langle\text{expr-int}\rangle \\
\langle\text{assign-real}\rangle &\rightarrow \langle\text{var-real}\rangle = \langle\text{expr-real}\rangle \\
\langle\text{expr-int}\rangle &\rightarrow \langle\text{expr-int}\rangle + \langle\text{term-int}\rangle \mid \langle\text{num-int}\rangle
\end{aligned}
$$

Contextual Dependencies

- All variables must be declared before use
- The same identifier may not be declared twice in the same block
- An array declared to have n dimensions must be referenced with n dimensions

Attribute Grammar

An attribute grammar $AG = (G, A, F)$ is defined by

- A CFG G
- For each terminal or nonterminal symbol X of G, a set of attributes $A(X)$, written $X.a$ if $a \in A(X)$
  - $A(X) = AS(X) \cup AI(x)$, with $AS(X) \cap AI(x) = \emptyset$
    - $AS(X)$ is the set of synthesized attributes of X: An attribute $a \in AS(X)$ is computed in rules where X is on the LHS
    - $AI(X)$ is the set of inherited attributes of X: An attribute $a \in AI(X)$ is computed in rules where X is on the RHS
- For each rule $X_0 \implies X_1 \ldots X_n$ of G, a set of semantic functions
  - $X_0.a \leftarrow f(A(X_1), \ldots, A(X_n))$ where $a \in AS(X_0)$
  - $X_i.a \leftarrow f(A(X_1), \ldots, A(X_n))$ where $1 \leq i \leq n$, $a \in AS(X_0)$

General Idea

- Symbols have attributes
- Rules have functions
- Synthesized attributes are calculated using the attribute values from the bottom of the tree
- Inherited attributes are calculated using the attribute values from the top of the tree
- We have our parse, and we enrich it with attributes!

Intrinsic Attributes

Values of synthesized attributes of some leaf nodes cannot be determined wiithin the tree.

Example

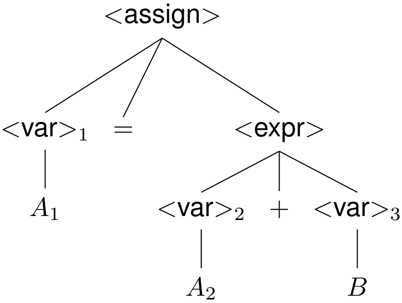
Consider that there is a variable A as a leaf node in the tree. If variables are not declared before use in that language (e.g. Fortran) initial character of the variable denotes its data type) then we must get its type from outside the tree (from the symbol table created during compilation).

Such attributes are called intrinsic attributes.

An attribute grammar

$$
\begin{aligned}
\langle \text{assign} \rangle &\rightarrow \langle \text{var} \rangle = \langle \text{expr} \rangle \\
\langle \text{expr} \rangle &\rightarrow \langle \text{var} \rangle + \langle \text{var} \rangle \\
\langle \text{var} \rangle &\rightarrow \mathrm{A} \mid \mathrm{B} \mid \mathrm{C}
\end{aligned}
$$

$A = A + B$



Type checking

- Get the types of A_1, A_2 and B from outside the tree.
- Synthesize (move up) type of $<$var$>_1$ from A_1, of $<$var$>_2$ from A_2, of $<$var$>_3$ from B
- Synthesize type of $<$expr$>$ from types of $<$var$>_2$ and $<$var$>_3$.
- Inherit (move down) type of $<$expr$>$ from $<$var$>_1$
- Compare the synthesized type and the inherited typeof $<$expr$>$

Use these attributes

Symbol	Attribute	Type
$<$var$>$	actual-type	synthesized
$<$expr$>$	actual-type	synthesized (type of RHS)
$<$expr$>$	expected-type	inherited (type of LHS)

Type Checking

$$\langle \text{assign} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expr} \rangle$$

$< expr > .expected\_type \leftarrow < var > .actual\_type$

$$\langle \text{expr} \rangle \rightarrow \langle \text{var} \rangle + \langle \text{var} \rangle$$

$< expr > .actual\_type \leftarrow$ if$(< var >_2 .actual_type =$ int) and
$< var >_3 .actual\_type =$ int then int else real endif
$< expr > .actual\_type =< expr > .expected\_type$

$$\langle \text{expr} \rangle \rightarrow \langle \text{var} \rangle$$

$< expr > .actual\_type \leftarrow < var > .actual\_type$
$< expr > .actual\_type =< expr > .expected\_type$

$$\langle \text{var} \rangle \rightarrow \mathrm{A} \mid \mathrm{B} \mid \mathrm{C}$$

$< var > .actual\_type \leftarrow$ lookup(<var>.string)

Let's demonstrate on the parse tree

Question

Create an attribute grammar that imposes the following constraints:

- Each variable must be declared before it is used
- A variable can appear only once in a declaration statement.

(int | double) id, id, id, ...

Answer: BNF Part

$$
\begin{array}{rcl}
\langle \text{decl} \rangle & \rightarrow & \langle \text{type} \rangle \langle \text{varlist} \rangle \\
\langle \text{type} \rangle & \rightarrow & \mathrm{int} \\
\langle \text{type} \rangle & \rightarrow & \mathrm{double} \\
\langle \text{varlist}_1 \rangle & \rightarrow & \langle \text{varlist}_2 \rangle , \langle \text{id} \rangle \\
\langle \text{varlist} \rangle & \rightarrow & \langle \text{id} \rangle
\end{array}
$$

Note: obviously, this is not a complete BNF. We are interested only in this part.

Answer: Attributes

Symbol	Attribute	Type
<type>	vartype	synthesized
<varlist>	vartype	inherited
<id>	vartype	inherited

Answer: Semantic functions

$$\langle \text{decl} \rangle \rightarrow \langle \text{type} \rangle \langle \text{varlist} \rangle$$

$< varlist > .vartype \leftarrow < type > .vartype$

$$\langle \text{type} \rangle \rightarrow \text{int}$$

$< type > .vartype \leftarrow \text{int}$

$$\langle \text{type} \rangle \rightarrow \text{double}$$

$< type > .vartype \leftarrow \text{double}$

$$\langle \text{varlist}_1 \rangle \rightarrow \langle \text{varlist}_2 \rangle , \langle \text{id} \rangle$$

$< varlist_2 > .vartype \leftarrow < varlist_1 > .vartype$
$< id > .vartype \leftarrow < varlist_2 > .vartype$

$$\langle \text{varlist} \rangle \rightarrow \langle \text{id} \rangle$$

$< id > .vartype \leftarrow < varlist > .vartype$

Notes

- The previous attribute grammar is not complete because we cover the propagation of types but not their use.
- The drawbacks of attribute grammars
  - It is difficult to build an attribute grammar. It increases the size of the CFG too much.
  - It is costly to compute the attribute values in a parse tree.
- One should ensure that the developed attribute grammar is the intended one!