

Semantics: the Meaning of Programs

I. Static Semantics: meaning of a program at compile time

A. Some BNF descriptions are not practical

1. E.g. type compatibility rules
2. E.g. all variables must be defined before they are referenced

B. Attribute Grammars

1. Synthesized & Inherited attributes
 - Intrinsic Attributes
2. Semantic functions: specifies assignments between attribute values
3. Predicate functions: specifies allowable relationships between attribute values
4. Attribute Grammar parse tree is a "decorated" BNF grammar parse tree

C. Examples of attribute grammars

1. Ada procedure "end" name matching proc. name:

Syntax rule: $\langle \text{proc_def} \rangle \rightarrow \text{procedure } \langle \text{proc_name} \rangle [1]$
 $\langle \text{proc_body} \rangle \text{ end } \langle \text{proc_name} \rangle [2]$

Semantic rule: $\langle \text{proc_name} [1] \rangle . \text{string} = \langle \text{proc_name} [2] \rangle . \text{string}$

2. Little Quilts

D. Example: Simple assignment statement

1. Syntax of grammar (rules) [overhead]
2. Attributes for non-terminals [overhead]
3. Attribute grammar for the simple assignment statement [overhead]
4. Decorating parse tree with attributes [overhead]. Order is important

E. Large (real) attribute grammars are complex and difficult to read and write and computationally expensive.

II. Dynamic Semantics: meaning of a program at run time

Useful for programmers, compiler writers, & correctness proofs

A. Operational Semantics

B. Axiomatic Semantics

1. Assertions: given the postcondition of a statement, we figure out what the precondition must be.
E.g. given $\text{sum} := 2 * x + 1$ & postcondition $\{\text{sum} > 1\}$, we work backwards and see that this will be true for $\{x > 5\}$
2. Weakest Preconditions: for the above example it is also true for $\{x > 50\}$, $\{x > 100\}$. The weakest (most general) precondition, however, is $\{x > 0\}$.
3. Proof process: starting at end of prog. (postcondition to last statement), work your way back establishing precondition to each statement. At each step the weakest precondition can be computed by one of the following:
 - a. axiom: a logical statement that is assumed to be true (e.g. the above example)
 - b. inference rule: a method of inferring the truth on one assertion on the basis of other true assertions (e.g. examples yet to come)
4. 4 kinds of statements must be handled and are explained further below. The first handled w/axiom, the others through inference rules
 - a. assignment statements (handled w/ an axiom)
 - b. sequences
 - c. selection
 - d. loops (logical pretest)
5. Assignment Statements
6. Sequences
7. Selection
8. Logical Pretest Loops

C. Denotational Semantics

III. Extending the Semantics of a Language

- A. An extension: a set of definitions which augment a language with an entirely new facility that can be used in the same way as existing facilities. E.g.:
 - 1. Original FORTRAN allowed variables to be defined, but not types or functions
 - 2. PASCAL, LISP: we can define a new object, a new function, or a new data type.
- B. In an *extensible* language, by building up a vocabulary of defined functions and/or procedures, we ultimately write programs in a language that is much more extensive and powerful than the bare language provided by the compiler. [F&G, p. 101]
- C. Historically: extensibility depends on uniform, general treatment of a language feature, as opposed to having a fixed set of keywords or symbols. E.g.
 - 1. BASIC not extensible: even var. names predefined (2 char. max)
 - 2. FORTRAN
 - a. had predefined math. fcn., but user could not write their own
 - b. Two data types "hard-wired": integer & real
 - 3. LISP: was extensible. Functions were considered basic, & user expected to write many of them.
 - 4. PASCAL
 - a. four primitive data types (bool, char, real, integer) can be combined to build new types
 - b. Still only certain combinations allowed: integer converted to reals in some cases.
This conversion relationship is not extensible to other types, as it is in C++
- D. What is involved in allowing semantic extension?
 - 1. E.g. ANSI C and C++ are semantic extensions of C
 - a. ANSI C added type checking for fcn. calls
 - b. C++ adds modules (classes), virtual functions, polymorphism
 - 2. To implement the above changes require:
 - a. modifying process to translate a function call
 - b. new info. added to symbol table
 - c. new restrictions on visibility (scope)
 - d. new type checking and type conversion algorithms
- E. Example: Semantic extension in FORTH.
 - 1. Two kinds of changes can be made:
 - a. Add new kinds of information to the symbol table
 - b. Modify the parser to translate new control structures
 - 2. FORTH words "CREATE" and "DOES>" denote a compiler extension
 - 3. It is interpreted
 - 4. The unextended version contains three semantic categories (data types): constant, variable, function
 - 5. Creating the array data type in FORTH (limited to a 2 x 3 array in this example)
[See overhead]