COEN 175

Lecture 6: Syntax-Directed Translation and Semantic Analysis

Syntax-Directed Translation

- **Syntax-directed translation** is the principle that we use the syntax, or structure, of the input program to guide its translation.
- Thus, the parser is “in charge”:
  - The parser calls the lexer when it needs a new token.
  - The parser will also invoke functions or actions to perform semantic analysis.
  - In a single-pass compiler, the parser would output the target language.
  - In a multi-pass compiler, the parser would construct the intermediate representation.

Attributes

- Each grammar rule can have an action that is executed when it is matched.
- Often these actions need to communicate information between them.
- This communication takes the form of **attributes**.
  - An attribute is simply any value or object that we associate with a grammar symbol.
  - For a grammar symbol *X*, we simply use to *X.name* to refer to the attribute *name.*
  - We don't communicate using global variables because many grammar rules are recursive.

Example: A Calculator

- As we parse an expression, suppose we want to calculate its value.
- We could use an attribute, called `val`, to hold the value of an expression.
- We will assume that the lexer sets the attribute for a number to be the value of the number.
- We will have to compute the value for additions, multiplications, etc.

Attribute Grammar

- Add attributes to the following grammar:

  $E \rightarrow E + T \mid T$
  $T \rightarrow T * F \mid F$
  $F \rightarrow (E) \mid \textbf{num}$

- The attribute grammar is simply:

$E \rightarrow E + T$		`{ E.val = E.val + T.val; }`
$\mid T$		`{ E.val = T.val; }`
$T \rightarrow T * F$		`{ T.val = T.val * F.val; }`
$\mid F$		`{ T.val = F.val; }`
$F \rightarrow (E)$		`{ F.val = E.val; }`
$\mid$ **num**		`{ F.val = num.val; }`

Kinds of Attributes

- If an attribute of a nonterminal is computed from the attributes of its children, it is **synthesized**.
  - In a recursive-descent parser, a synthesized attribute is the return value of a function.
  - Our calculator used synthesized attributes exclusively.
- If an attribute of a symbol is computed from the attribute of its parent (or ancestors), it is **inherited**.
  - In a recursive-descent parser, an inherited attribute is a parameter to a function.
  - A declaration in C would use inherited attributes since the type specifier must be passed down into the declarator.

Abstract Syntax Trees

- A parse tree is also called a concrete syntax tree.
  - A parse tree has every token in the input program and every production used, many of which are not needed.
  - For example, expression grammars often have long chains of productions used only to enforce precedence.
- An **abstract syntax tree**, or AST, omits any information that we don't need in later analysis.
  - An AST is a common intermediate representation.
  - There is no one, definitive AST. It is a design choice as to which symbols are included.
  - Although, there are some common conventions, especially using **expression trees** for expressions.

Semantic Analysis

- **Semantic analysis** is the process of determining the meaning of the well-formed input program.
  - Most of semantic analysis is **semantic checking**.
- Semantic checks are either **static** or **dynamic**:
  - "Static": compile-time (things aren't changing)
  - "Dynamic": run-time (things are changing)
- Checking that a variable is declared before being used is a static semantic check.
- Checking that an array index is within the bounds of the array is a dynamic semantic check.

Static Semantic Checks

- Uniqueness checks: verifying that there is one and only one definition of an object or identifier.
  - Is a variable declared before being used?
  - This is the third phase of the project.
- Type checks: verifying that the operands to an operator are of the correct type.
  - Can you add a pointer and an integer?
  - This is the fourth phase of the project.
- Control checks: verifying that the control structure of a program is consistent.
  - Is a `continue` inside a loop?