CS143
Compilers

Lecture 6

July 23, 2001

Syntax-Directed Translation

- Context-free grammars are useful not only for checking syntax, but as a guide in translation.
- In syntax-directed translation, we associate *attributes* to each grammar symbol. The value of each attribute is computed using *semantic rules*, which are associated with each production of the grammar.
- There are two notations used to associate semantic rules with productions: *syntax-directed definitions* and *translation schemes*.

Evaluation of Semantic Rules

- Regardless of the notation used, evaluation of semantic rules proceeds by building a parse tree for the input, and then traversing the tree in some order so that the rules can be evaluated.
- Such rules can be used to perform a variety of tasks, including code generation, storing information in a symbol table, and checking semantic validity.
- In many cases, the parse tree can be built and the rules can be evaluated within a single pass.

Syntax-Directed Definitions

- A *syntax-directed definition* is a context-free grammar in which each grammar symbol is associated with a set of attributes.
- Attributes can be of any type, to represent any information that is needed.
- Given a node in a parse tree, the value of the attribute for the grammar symbol at that node is computed using attribute values of neighboring nodes.
- The process of computing attribute values at each node is called *annotating* or *decorating* the parse tree.

Synthesized Attributes

- Synthesized attributes are the most commonly used type of attributes in practice.
- The value of a synthesized attribute at a given node in the parse tree is computed using only the values of attributes from the node's children.
- A syntax-directed definition that uses only synthesized attributes is an *S-attributed definition*. A parse tree for such a definition can be annotated from the bottom up, which is why an LR parser can easily be used to efficiently implement such a definition.

Inherited Attributes

- The value of an inherited attribute at a given node in a parse tree is computed using attribute values from the siblings and/or parent of the node.
- A common use of an inherited attribute is to keep track of whether an identifier appears on the left or right side of an assignment, in order to know whether its address or value is needed.
- A syntax-directed definition can always be written so as to use only synthesized attributes, but this can be more cumbersome than using inherited attributes.

Dependency Graphs

- A dependency graph is used to help determine the order in which semantic rules are to be evaluated within a given parse tree.
- Each attribute is assigned a node in the graph, and edges are assigned based on the dependencies of each attribute on other attribute.
- A topological sort of the dependency graph yields the order of evaluation of attribute values.
- This order can be computed either at compile time, or at compiler-construction time.

Bottom-up Evaluation of S-Attributed Definitions

- Now that we have used a syntax-directed definition to assign semantic rules to our grammar, we are ready to build a translator based on this definition.
- In general, building such a translator is very difficult, but not so for certain classes of definitions, including S-attributed definitions.
- A bottom-up parser facilitates evaluation of attributes for such definitions.

Synthesized Attributes on the Parser Stack

- Attribute values can be maintained on the stack of a bottom-up parser. During a reduction, the attribute value of the new non-terminal being placed on the stack is computed from the attribute values contained in the handle.
- In some cases, we can compute attributes from values that are located deeper within the stack. This allows us to include some inherited attributes in our definition, though care should be taken in using this approach.

L-Attributed Definitions

- When translation occurs during parsing, the order in which attributes are evaulated is constrained by the order in which nodes of the parse tree are "created".
- An L-attributed definition is a definition in which the attributes can be evaluated by traversing the parse tree depth-first, ordering child nodes from left to right.
- Any syntax-directed definition based on an LL(1) grammar is L-attributed, as are many that are based on LR(1) grammars.

L-Attributed Definitons, Cont'd

- Formally, a syntax-directed definition is L-attributed if, for each production $A \rightarrow X_1 X_2 \ldots X_n$ the inherited attributes of any symbol X_j depend only on:
  - The inherited attributes of A, and
  - The attributes of $X_1, \ldots, X_{j-1}$.
- Every S-attributed definition is also L-attributed, since the above restrictions only apply to inherited attributes.

Translation Schemes

- A translation scheme is a generalization of a syntax-directed definition. It includes the precise order in which semantic rules are evaluated, relative to the addition to the parse tree of nodes associated with grammar symbols on the right side of each production.
- Parser generators such as yacc and bison are used to specify translation schemes.

Translating Synthesized Attributes

- A translation scheme is easy to design from an S-attributed definition. Given a production $A \rightarrow \alpha$, the synthesized attributes of A can be computed by adding a semantic action after α in the translation scheme.
- When the reduction from α to A takes place, the semantic rule is evaluated, using the attribute values of each grammar symbol in α. Placing the rule after α in the translation scheme indicates that the rule is not to be evaluated until each symbol in α is available on the parsing stack. Since the attributes are synthesized, we can guarantee that the attribute values for each symbol in α are also available.

Translating Inherited Attributes

Evaluating inherited attributes is not so simple. The following restrictions must be observed:

- An action cannot refer to the attribute value of a symbol that occurs to the right of the action in a production.
- An inherited attribute for a symbol within a right side must be computed in an action to the left of that symbol.
- The synthesized attributes of a non-terminal on the left side can only be computed after all of its dependencies have been computed.

Top-Down Translation

- When implementing a translation scheme using a predictive parser, one must ensure that the scheme, as well as the grammar, are suitable for predictive parsing.
- This requires removing left recursion from the translation scheme, as well as the grammar.
- Removing left recursion from a grammar requires the introduction of new non-terminals. Semantic actions associated with these new non-terminals help to create the modified translation scheme.
- When using recursive descent, inherited attributes of a non-terminal are passed as arguments to the function for that non-terminal.

Bottom-up Evaluation of Inherited Attributes

- L-attributed definitions are frequenly implemented within a bottom-up parser.
- Such definitions, including those containing inherited attributes, involve semantic actions that are *embedded*, i.e. they are evaluated while the right side of a production is being recognized, instead of when the reduction takes place.
- Such embedded actions, though convenient, can restrict the ability of the parser to make parsing decisions. In particular, it can introduce conflicts into a left-recursive grammar.

Removing Embedding Actions from Translation Schemes

- Implementing bottom-up evaluation involves, at least conceptually, augmenting the grammar so that all semantic actions occur at the end of the right side of a production.
- This is accomplished by adding new *marker* non-terminals that derive ε. They are inserted into existing productions in place of embedded actions.
- These embedded actions are moved to the right sides of the ε-productions for these marker non-terminals, so that they are always evaluated on some reduction.

Inheriting Attributes on the Parser Stack

- Given a production $A \rightarrow XY$, suppose that X has a synthesized attribute s.
- Since s is computed and stored on the parsing stack before any reductions take place in the subtree below Y, it follows that s can be inherited by Y, and in turn propagated to other symbols in the subtree rooted at Y.
- For this to work, it must be verified that when parsing the right side of a production $Y \rightarrow \alpha$, the handle α is always preceded by X on the parsing stack.

Replacing Inherited by Synthesized Attributes

- In many cases, one can avoid the use of inherited attributes by changing the grammar.
- One rule of thumb is as follows: if a non-terminal A has an inherited attribute and is on the left side of left-recursive productions, the attribute may be converted into a synthesized attribute by using right recursion instead.
- Note that changing from left recursion to right recursion changes associativity, so this heuristic should only be used with non-associative constructs.

Space for Attribute Values

- Space for attribute values can be assigned either during compilation, or it can be pre-assigned during compiler-construction time.
- In either case, it is important to consider the lifetime of an attribute. Its lifetime ends when no other attributes depend on its value.
- It is advisable to avoid making explicit copies of attributes, in order to save space.
- When assigning space at compiler-construction time, often multiple stacks are used, where stacks are associated with nodes in a parse tree. If attribute lifetimes do not overlap, they can share a stack.