Semantics: Context-Sensitive Analysis

Questions not answerable by a CFG

- Is x a scalar, an array, or a function?
- Is x declared before it is used?
- Are any names declared but not used?
- Which declaration of x does this reference?
- Is an expression type-consistent?
- Does the dimension of a reference match the declaration?
- Where can x be stored? (heap, stack, …)
- Is x defined before it is used?
- Is an array reference in bounds?
- Does function foo produce a constant value?

Context-Sensitive Analysis

- Why is it hard?
  - Need non-local information
  - Answers depend on values, not on syntax

Solutions –

Attribute grammars – augment CFG with rules, calculate attributes for grammar symbols

ad hoc techniques – augment grammar with arbitrary code, execute at corresponding reduction, store information in attributes, symbol tables

Attribute Grammars

- Generalization of context-free gammars
- Each grammar symbol has an associated set of attributes
- Augment grammar with rules that define values
  - Not allowed to refer to any variables or attributes outside the current production
- High-level specification, independent of evaluation scheme

Attribute Types

- Values are computed from constants and other types
  - Synthesized attribute – value computed from children
  - Inherited attribute – value computed from siblings, parent, and own attributes

Attribute flow

- S-attributed grammar
  - Uses only synthesized types
  - Bottom-up attribute flow
- L-attributed grammar
  - Attributes can be evaluated in a single left-to-right pass over the input
    - Each synthesized attribute of LHS depends only on that symbol's own inherited attributes or on attributes (synthesized or inherited) of the production's RHS symbols
    - Each inherited attribute of a RHS symbol depends only on inherited attributes of the LHS symbol or on attributes (synthesized or inherited) of symbols to its left in the RHS

Problems with Attribute Grammars

- Handling non-local information
- Storage management
- Syntax tree traversal to extract information
- Tools for automation

Action Routines

- Ad hoc translation scheme (attribute evaluator) that is interleaved with parsing
  - Based on the idea behind rule-based evaluators for attribute grammars
  - Attribute flow constrained to a single direction, either synthesized or inherited (e.g., L-attributed)
  - Also called syntax-directed translation
- Allow arbitrary actions
- Provide central repository
- Can place actions amid productions

Typical uses – build abstract syntax tree, symbol table, perform error/type checking

Top-Down Evaluation

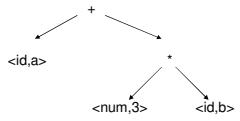
- Can associate storage with nodes in the parse tree (also for bottom-up)
- Inherited attributes parameters to parsing routine, synthesized attributes return values
- Automatic management also possible with separate parse and attribute stack
  - Action routines interspersed with RHS
  - Problem: many copies
  - Solution: Ad hoc management by explicitly pushing and popping attributes

Bottom-Up Evaluation

- Attribute stack that mirrors the parse stack
- S-attributed grammar – perform action at the time of reduction
- L-attributed grammar – possible but not always
  - No obvious place to store inherited attributes (don't know what you're inheriting from)
  - Use marker symbols (semantic hooks) to know depth of symbol from which you are inheriting
  - Can put marker symbol in the TRAILING PART (production uniquely determined) but not in the LEFT CORNER of an RHS

Abstract Syntax Tree

- An abstract syntax tree is the procedure's parse tree with the nodes for most non-terminal symbols removed
  - E.g., "a + 3 * b"



Symbol Tables

- Associates values or attributes (e.g., types) with names
  - Names
    - Variable and procedure names
    - Literal constants and strings
  - Attributes
    - Textual name
    - Data type
    - Declaring procedure
    - Lexical level of declaration
    - If array, number and size of dimensions
    - If procedure, number and type of parameters

Symbol Table Implementation

- Usually implemented as hash tables
- Return closest lexical declaration to handle nested lexical scoping
- Solution used in your project
  - Use one symbol table per scope
  - Chain tables to enclosing scope
  - Insert names in tables for current scope
  - Start name lookup in current table, checking enclosing scopes in order if needed