Attribute grammars

- Context-free grammars are powerful notations for compiling
- At the same time they are, indeed, context-free
  - For example, they can recognize strings such as $x^n y^n$ but not $x^n y^n z^n$
  - In general, CFGs are limited in the kind of underlying computations they can represent
- Attribute grammars (Knuth 1968) are a formal approach to overcome such limitations by augmenting a CFG with attributes and equations to compute those attributes

Example (Aiken, Berkeley)

- $E ::= E' + E | E'$
- $E' ::= \text{int} * E' | \text{int}$

- What if not only want to represent the expressions as a syntax tree, but we also want to compute their result?
- Augment terminals and non-terminals with attributes
- Augment productions with equations

The attribute grammar

- $E ::= E' + E1$  
  $E\cdot\text{val} = E'\cdot\text{val} + E1\cdot\text{val}$
- $E ::= E'$  
  $E\cdot\text{val} = E'\cdot\text{val}$
- $E' ::= \text{int} * E1'$  
  $E'\cdot\text{val} = \text{int}\cdot\text{val} * E1'\cdot\text{val}$
- $E' ::= \text{int}$  
  $E'\cdot\text{val} = \text{int}\cdot\text{val}$

- All attributes are integer (in this example), referred to by $a\cdot\text{val}$ where $a$ is a symbol in the grammar
- For terminal symbols, the attribute’s value is defined to be the lexeme (as returned by the scanner)
- For non-terminal symbols, the attribute’s value is defined by the associated equation
- In this case, the final value of $E\cdot\text{val}$ is supposed to be the value of the parsed expression

Miscellaneous

- The attribute of some symbols is unused
- Fresh attributes are associated with every node in the parse tree – that instances of grammar symbols have their own attribute value
- The semantic actions specify a system of equations; they don’t say in what order the equations are resolved.
  - Side-effects in equations may require an understanding of the order in which attributes get computed
  - In the example, the $\text{val}$ attribute can be evaluated bottom-up: this is not always true
Two kinds of attributes

- Synthesized: attribute value depends on descendants of the node
  - Example: the `val` attribute above
- Inherited: attribute value depends on parent and siblings of the node
  - Example: symbol table environment – why might we want this?

Reprise

- Attribute grammars can allow the parsing of richer languages (e.g., $x^y z^i$ can be parsed by adding equations that count how many of each terminal are in a sequence and making sure that they match)
  - These are usually more constrained languages – for example, ensuring that a syntactically legal program also satisfies the typing restrictions
- They can also associate meaning to grammars
  - When a parser tree is passed to semantic analysis, a lot of information is taken for granted
  - Example: $3 \times 4 = 12$

Compiling for multicore

- Multi-core is here
- Why does this place fear in the heart of compiler writers?
- Who else does it scare?
- Why?

Issues

- Concurrency is hard(er)
- Compile concurrency or infer concurrency or both?
- Homogeneous vs. heterogeneous
  - Processors, access times, etc.
- What layer should provide/exploit the concurrency?
  - Architecture, language, middle-ware, application, etc.?