Applications of Context Free Languages

Mark Greenstreet, CpSc 421, Term 1, 2006/07

Lecture Outline

Context Free Languages

- Parsing ant Interpretation
- Ambiguity

Parsing (1/2)

- Given a CFG, G, we can write a program that reads a string, and if the string is in $L(G)$, produces the parse-tree for the derivation of the string.
  - There's an $O(n^3)$ algorithm that handles any grammar – it's mostly of theoretical interest.
  - Recursive descent parsers handle the non-determinism by trying each possibility in turn, and backtracking. Although this is worst-case exponential time, recursive descent works quite well for the grammars of real programming languages.
  - There are automatic parser generators that produce table driven parsers. These only work with a subset of CFGs (typically LALR(1) grammars), but this subset is sufficient for nearly all practical applications.

Parsing (2/2)

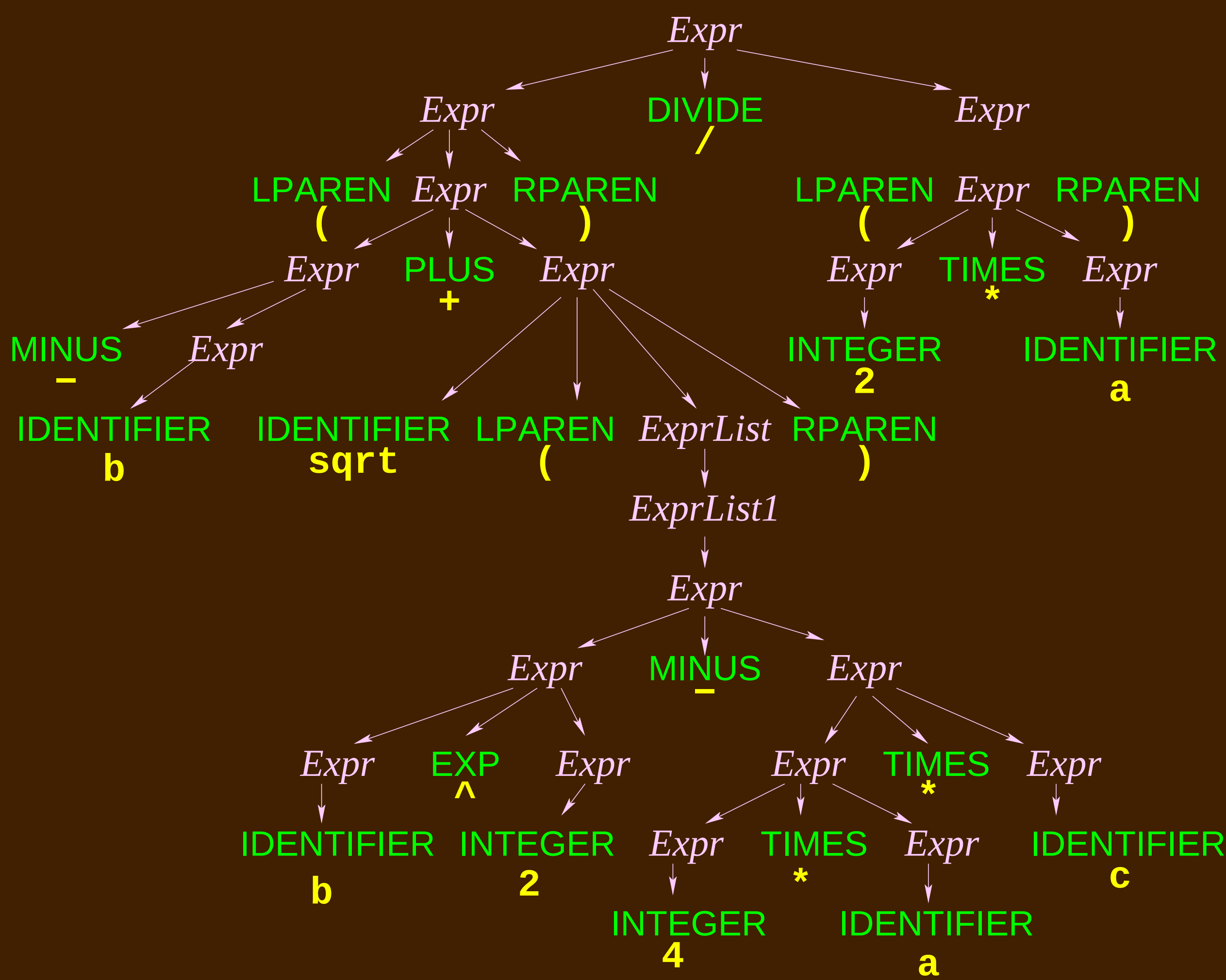
- With parser generators, writing a parser is nearly as easy as writing down the CFG.
  - The “nearly” part is because or the restrictions on the grammar mentioned above.
  - If your grammar violates these restrictions, you can adjust the details, but this requires some understanding of CFGs – that’s (one reason) why you’re in this class

Interpretation

- Once you have a parse tree, interpretation is “easy”.
  - For each terminal, determine the value for that terminal.
    - Example: `INTEGER`. Take the string for this particular integer, e.g. `17`, and convert it to a number.
    - Example: `IDENTIFIER`. Maintain a hash table that maps names of variables (i.e. `IDENTIFIER`s) to their values. Get the value for this variable from the hash table.
  - For each terminal, write an interpretation function. This function takes the values of the child nodes for this parse tree node, and computes a value for the node itself.
    - Example: $Expr\ \ rightarrow\ \ Expr_1$ `PLUS` $Expr_2$.
      The parse-tree node is for an $Expr$. It’s children are $Expr_1$ and $Expr_2$.
      - Invoke the evaluation methods for each of these child expressions to get their values.
      - Compute the sum of these two values.
      - Set the value for this node to the sum.

Example:

$$(-b + sqrt(b^2 - 4 * a * c))/(2 * a)$$



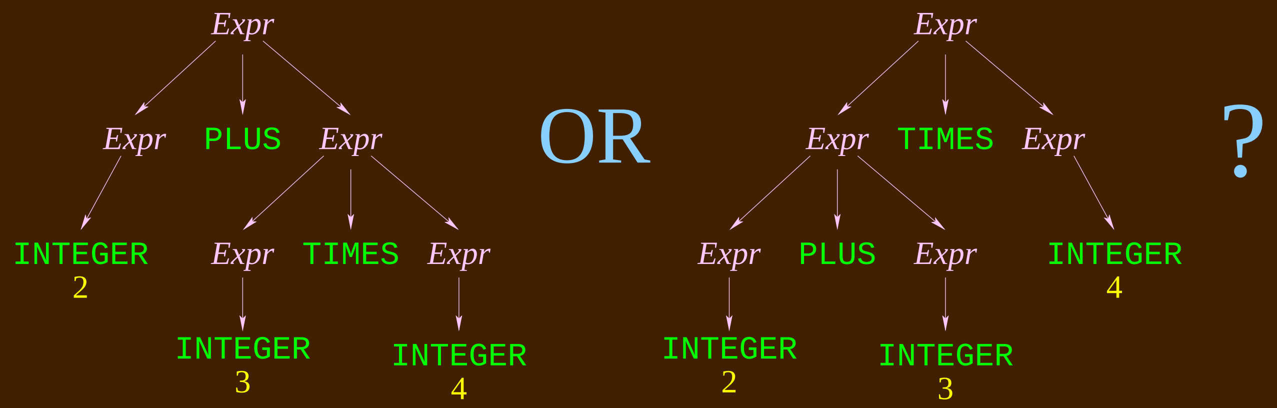
More Interpretation

- $whileStatement \rightarrow$ LPAREN $Expr$ RPAREN $Statement$
- What the interperter does:
  - Evaluate $Expr$.
  - If the result is `false`, done.
  - Otherwise, evaluate $Statement$; then, go back and test $Expr$ again, and continue.

Ambiguity

$$Expr \rightarrow Expr\ \texttt{PLUS}\ Expr \mid Expr\ \texttt{TIMES}\ Expr$$
$$\mid \texttt{INTEGER}$$

Consider: `2 + 3 * 4`



Unambiguous Arithmetic

$$\begin{array}{rcl} \mathit{Term} & \rightarrow & \mathit{INTEGER} \mid \texttt{IDENTIFIER} \\ \mathit{Product} & \rightarrow & \mathit{Term} \mid \mathit{Product}\ \texttt{TIMES}\ \texttt{Term} \\ \mathit{Sum} & \rightarrow & \mathit{Product} \mid \mathit{Sum}\ \texttt{Plus}\ \mathit{Product} \\ \mathit{Expr} & \rightarrow & \mathit{Sum} \end{array}$$