Operator Precedence

Operator Precedence

- *Operator grammar*
  - no right side of a production contains adjacent non-terminals
- S ⇒ EoE | id ,
  o ⇒ + | - | * | / (X)
- If a grammar is an operator grammar and it has no productions with null of the RHS, then there is a operator-precedence parser for that grammar
- Special case of a *shift-reduce* parser

Precedence Grammars

- Parse with shift/reduce
- No production right side is ε, and no right side has two adjacent nonterminals
- bad
  - multi-precedence operator (-) difficult
  - can't be sure parser matches the grammar!
  - only works for some grammars
- good — simple, simple, simple
- Build on non-reflexive *precedence relations* that we denote as $\cdot>$, $\doteq$, $<\cdot$ (typographical convenience for dotted forms of <,=,> as in text)

Computing Precedence

- Precedence is disjoint. Can have

  `a <· b`

  `a <· b` and `a ·> b`

  `c <· b, c ≅ b, c ·> b`

  is read "yields precedence" or "equal precedence"

- Obtain precedence by manual assignment using traditional associative and precedence rules, or mechanically from nonambiguous grammar

- How to process
  - "Ignore" nonterminals, and then delimit handle from right side ·> and then back up to the left side <·

Operator Precedence Parser

- Remove (hide) nonterminals and place precedence relationship between pairs of terminals
  - (1) Represent `id + id * id` as
    `$ <· id ·> + <· id ·> * <· id ·> $`
  - (2) Apply scanning method
    - a) scan from left toward right to find ·>
    - b) backwards scan to <· or $
    - c) handle is located between start and end of scan
    - d) reduce the handle to the corresponding nonterminal
- Relies on the grammar's special form
  - (1) In grammar rule, no adjacent nonterminals on the right hand-side (by definition), so no right sentential form will have two non-terminals
  - (2) Form is $\beta_0 a_1 \beta_1 \cdots a_n \beta_n$

    β_i is nonterminal or ε

    a_i is a nonterminal