Operator Precedence

Introduction

- For a certain small class of grammars we can construct efficient shift reduce parsers by hand
- These grammars have the property that
  - no production right side is $\in$
  - or has two adjacent nonterminals
- Such a grammar is called an operator grammar
- It is used to parse expressions

Disadvantages

- hard to handle the –minus sign since it has two precedence (unary or binary)
- relationship between the grammar and the operator precedence parser is weak (therefore the language accepted may not be of the grammar)
- only a small class of grammars can be parsed

Precedence Relations

- In operator-precedence parsing we use
- Three disjoint precedence relations ($\lessdot$, $\doteq$, $\gtrdot$)
  - These precedence relations guide the selection of handles
- a $\lessdot$ b where a yield precedence to b
- b $\gtrdot$ a where b takes precedence over a
- a $\doteq$ b where a has same precedence as b
- Or none of $\lessdot$, $\doteq$, $\gtrdot$

Precedence Relations

- There are two common ways of determining what precedence relation should hold between a pair of terminals

1. Traditional notion of associativity and precedence of operators
   - Example

     If * has higher precedence than + then we make

     $+ \lessdot * \text{ and } * \gtrdot +$
2. Second method to disambiguate the grammar in such a way that it will reflect the associativity and precedence in its parse trees

Using Operator Precedence Relations

- delimit the handle of a right sentential form with
- $\lessdot$ marking the left end
- $\doteq$ appearing in the interior of the handle (if any)
- $\gtrdot$ marking the right end
- Consider an operator grammar where there are no right sentential form having two adjacent nonterminals
- We write the sentential form as
- $\beta_0 a_1 \beta_1 \ldots a_n \beta_n$
- where each β_l is either $\in$ or a single nonterminal
- and each a_i is a single terminal

- Between a_i and a_{i+1} exactly one of the relations ($\lessdot$, $\doteq$, $\gtrdot$) holds
- $ marks each end of the string
- Define $ $\lessdot$ b and b $\gtrdot$ $ for all terminals b

Operator precedence relation for the right sentential form id + id * id

	id	+	*	$
id		⋗	⋗	⋗
+	⋖	⋗	⋖	⋗
*	⋖	⋗	⋗	⋗
$	⋖	⋖	⋖	

- The string with the precedence relations inserted is
- $ ⋖ id ⋗ + ⋖id ⋗ * ⋖ id ⋗ $
- The handle can be found by the following process

1. scan the string from the left end until the left most ⋗ is encountered
2. then scan backwards(to the left) over any ≐'s until a ⋖ is encountered
3. The handle contains everything to the left of the first ⋗ and to the right of ⋖ including any intervening or surrounding nonterminal

- Here the handle is the first **id**
- **And id can be reduce to E and the right sentential form becomes**

  E + id *id

- **If the precedence relation $\lessdot$ or $\doteq$ holds** between the topmost terminal symbol of the stack and the next input symbol, the **parser shifts** (parser has not found the right end of the handle)
- **If the $\gtrdot$ holds**, a **reduction** is called for
- **If no precedence relation holds between a pair of terminals**, then a **syntactic error** has been detected.

Operator Precedence Relations from associativity and Precedence

- Creating Operator precedence in this way requires us to follow certain rules
- The Rule are designed to select the “proper” handles to reflect a given set of associativity and precedence rules for binary operators

1. If operator θ_1 has higher precedence than θ_2 then
   - make $\theta_1 \gtrdot \theta_2$ and $\theta_2 \lessdot \theta_1$
   - E.g., if * has higher precedence than +
     - * $\gtrdot$ + and + $\lessdot$ *
     - Ensure that in an expression E + E * E +E, the central E*E is reduced first

- ◦

2. If operator θ_1 and θ_2 have same precedence
   - make $\theta_1 \gtrdot \theta_2$ and $\theta_1 \gtrdot \theta_2$ if operators are left associative
   - Or make $\theta_1 \lessdot \theta_2$ and $\theta_1 \lessdot \theta_2$ if operators are right associative

   - E.g., **if + and – are left associative**
     - Make + $\gtrdot$ - and - $\gtrdot$ +
   - **If right associative**
     - + $\lessdot$ - and - $\lessdot$ +
   - These ensure that in an expression such as E-E+E will have E-E as a handle
   - **If $\uparrow$ is right associative then**
     - $\uparrow \lessdot \uparrow$
   - And E $\uparrow$ E$\uparrow$E will have the last E $\uparrow$E selected as a handle

3. Make θ ⋖id, id ⋗ θ, θ ⋖(, (⋖ θ,) ⋗ θ, θ ⋗), θ ⋗ $and $ ⋖ θ for all operators θ
   - Let

(≐)	$ ⋖(	$ ⋖ id
(⋖(	id ⋗ $	) ⋗ $
(⋖ id	id ⋗)	) ⋗)

Operator precedence relations

- ↑ is of highest precedence and right associative
- * and / are of next highest precedence and left associative
- + and – are of lowest precedence and left associative

	+	–	*	/	↑	id	(	)	$
+	·>	·>	<·	<·	<·	<·	<·	·>	·>
–	·>	·>	<·	<·	<·	<·	<·	·>	·>
*	·>	·>	·>	·>	<·	<·	<·	·>	·>
/	·>	·>	·>	·>	<·	<·	<·	·>	·>
↑	·>	·>	·>	·>	<·	<·	<·	·>	·>
id	·>	·>	·>	·>	·>			·>	·>
(	<·	<·	<·	<·	<·	<·	<·	=	
)	·>	·>	·>	·>	·>			·>	·>
$	<·	<·	<·	<·	<·	<·	<·		

Fig. 5.7. Operator precedence relations.

Operator precedence Grammars

- Let G be an $\in$ free operator grammar
- For each two terminal symbols a and b

1. **a $\doteq$ b** if there is a right side of a production of the form $\alpha a \beta b \gamma$ where β is either $\in$ or a single nonterminal
   - a $\doteq$ b if a appears immediately to the left of b in a right side, or if they appear separated by one nonterminal
   - E.g., S$\rightarrow$iCtSeS $\mathrm{implies\ that}$ i $\doteq$ t and t $\doteq$ e

OPR

1. **a ⋖ b** if for some nonterminal A there is a right side of the form $\alpha a A \beta$ and $A \Rightarrow^{+} \gamma\, b\delta$, where γ is either $\in$ or a single nonterminal
   - a ⋖ b if a nonterminal A appears immediately to the right of a and derives a string in which b is the first terminal symbol
   - The derivation is one step
   - E.g., $S \rightarrow iCtS$, and $C \Rightarrow b$, so $i \lessdot b$
2. **a ⋗b** if for some nonterminal A there is a right side of the form $\alpha A b \beta$, and $A \Rightarrow^{+} \gamma\, a\delta$ where δ is either $\in$ or a single nonterminal
   - a ⋗b if a nonterminal appearing immediately to the left of b derives a string whose last terminal is a
   - E.g., $S \rightarrow iCtS$ and $C \Rightarrow b$ imply $b \gtrdot t$

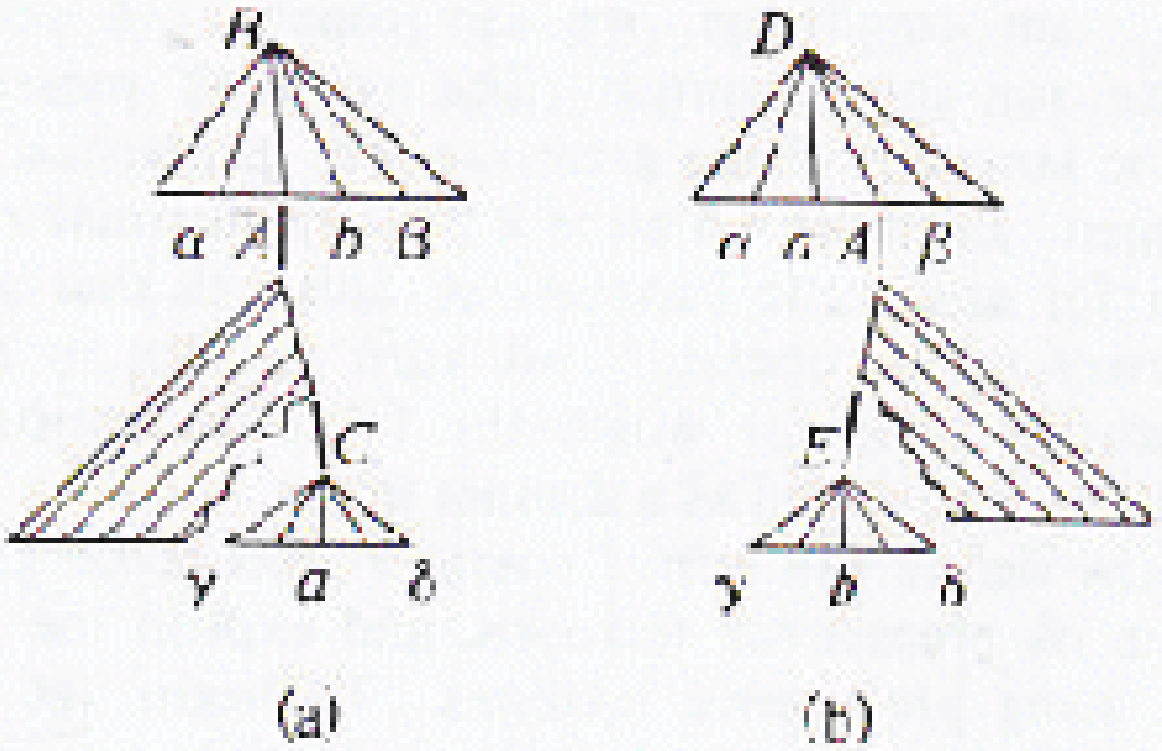

Fig. 5.8. Parse trees indicating $a \cdot> b$ and $a <\cdot b$.

OPR

- Definition:
  - An operator precedence grammar is an $\in$ free operator grammar in which the pr4ecedence relations $\lessdot$, $\doteq$, $\gtrdot$ constructed are disjoint. That is any pair of terminals a and b, never more than one of the relations a $\lessdot$ b, a $\doteq$ b,a $\gtrdot$ b is true

Identifying First and Last terminals

E→E + T | T

T →T * F | F

F →(E)| id

- F ⇒(E), F ⇒id, here (and id are the first symbols
- T⇒T * F, here * could be the first and last symbol
  - And F ⇒(E), F ⇒id, so (and id could also be the first symbols
- Hence the first symbols derivable form T are (, *, id

First and Last Terminals

Non terminals	**First terminals**	**Last terminals**
E	*,+,(,id	*,+,),id
T	*,(,id	*,), id
F	(,id	),id

	+	*	(	)	id	$
+	$\cdot>$	$<\cdot$	$<\cdot$	$\cdot>$	$<\cdot$	$\cdot>$
*	$\cdot>$	$\cdot>$	$<\cdot$	$\cdot>$	$<\cdot$	$\cdot>$
(	$<\cdot$	$<\cdot$	$<\cdot$	$\doteq$	$<\cdot$	
)	$\cdot>$	$\cdot>$		$\cdot>$		$\cdot>$
id	$\cdot>$	$\cdot>$		$\cdot>$		$\cdot>$
$	$<\cdot$	$<\cdot$	$<\cdot$		$<\cdot$	

Fig. 5.10. Operator-precedence relations.

Leadings and Trailings Sets

- Two sets LEADING(A) and TRAILING(A) for each nonterminal A is define by:

1. LEADING(A)={a|A $\Rightarrow^{+} \gamma$ aδ where γ is either $\in$ or a single nonterminal}
2. TRAILING(A)=(a|A $\Rightarrow^{+} \gamma$ aδ where δ is either $\in$ or a single nonterminal}

- Determine the $\lessdot$ relation by looking in each right side for adjacent symbols …aA…, then a is related by $\lessdot$ to each b in LEADING(A)
- The TRAILING(A) helps to compute the $\gtrdot$ relations

Computation of LEADING

- Two Rules

1. A is in LEADING(A) if there is a production of the form $A \rightarrow \gamma\ a\delta$ where γ is either $\in$ or a single nonterminal
2. If a is in LEADING(B), and there is a production of the form $A \rightarrow B\alpha$, then a is in LEADING(A)

- Construct a Boolean array L[A,a] where L[A,a] is set to true iff terminal a is in LEADING(A), for nonterminal A
  - L[A,a] is initialize to true iff rule 1 follows
- Keep a stack of pairs (B,a) such that L[B,a] has been set to true, but the pair (B,a) has not been used in rule 2 to find a new pairs (A,a) such that a is in LEADING(A)

Leading and Trailing sets

- Compute LEADING sets
- Compute TRAILING sets
- Computation of operator precedence relations using LEADING and TRAILING Sets

Operator Precedence Parsing algorithm

- The precedence relations from some operator - precedence grammar and an input string of terminals from that grammar
- Output: No Output- however a parse tree can be constructed as we parse, with one nonterminal labeling all interior nodes
- Method: Let the input string be $a_1a_2…a_n\$$

Initially the stack contain $

Operator Precedence Parsing algorithm

1. **Repeat forever**
2. **If** only $ is on the stack and only $ is on the input **then** accept and break

   Else

   Begin
3. Let *a* be the topmost terminal symbol on the stack and let *b* be the current input symbol;
4. **If** $a \lessdot b$ or $a \doteq b$ **then** shift *b* onto the stack
5. **Else** if $a \gtrdot b$ **then** /* reduce */
6. **Repeat** pop the stack
7. **Until** the top stack terminal is related by $\lessdot$ to the terminal most recently popped
8. **Else** call the error correcting routine

end

Creating parse tree

- Create a node for each terminal shifted onto the stack at line 4
- Then when the loop of lines 6-7 reduces by some production we create a node whose children are the nodes corresponding to whatever is popped off the stack
- After line 7 we place on the stack a pointer to the node created
  - Some of the symbols popped by line 6 will be pointers to nodes
- The comparison of line 7 continues to be made between terminals only
- Pointers are popped with no comparison being made

Precedence Functions

- Compilers using operator precedence parsers need not store the table of precedence relations.
- The table can be encoded by two precedence functions **f** and **g,** which map terminals a and b

1. **f(a) $\lessdot$ g(b) whenever a $\lessdot$ b**
2. **f(a) $\doteq$ g(b) whenever a $\doteq$ b**
3. **f(a) $\gtrdot$ g(b) whenever a $\gtrdot$ b**

- Thus the precedence relation between a and b can be determined by a numerical comparison between f(a) and g(b)

Method for finding Precedence Functions for a table

1. Create symbols f_a and g_b for each a that is a terminal or $
2. Partition the created symbols to as many groups as possible in such a way that if a $\doteq$ b then f_a and g_b are in the same group
   - We may have to put symbols in the same group even if they are not related by $\doteq$
   - E.g., if a $\doteq$ b and c $\doteq$ b then f_a and f_c must be in the same group since they are in the same group as g_b

3. Create a directed graph whose nodes are the groups found in 2.
    1. For any a and b, if a $\lessdot$ b, place an edge from the group of g_b to the group of f_a
    2. If a $\gtrdot$ b place an edge from group of f_a to that of g_b
4. If the graph constructed in 3 has a cycle then no precedence function exist
    - If there are no cycle then let
        1. f(a) be the length of the longest path beginning at the group of f_a
        2. g(a) be the length of the longest path from the group of g_a

example

- Consider the matrix
- The graph representing precedence functions