Operator-Precedence Parser

Akshaya, Govt. Engg College Trivandrum

- **Operator grammar**
  - small, but an important class of grammars
  - we may have an efficient operator precedence parser (a shift-reduce parser) for an operator grammar.
- In an *operator grammar*, no production rule can have:
  - ε at the right side
  - two adjacent non-terminals at the right side.

- Ex:

  E→AB E→EOE E→E+E |
  A→a E→id E*E |
  B→b O→+ | * | / E/E | id

not operator grammar not operator grammar operator grammar

Precedence Relations

- In operator-precedence parsing, we define three disjoint precedence relations between certain pairs of terminals.

  $a \lessdot b$ b has higher precedence than a

  $a \doteq b$ b has same precedence as a

  $a \gtrdot b$ b has lower precedence than a

- The determination of correct precedence relations between terminals are based on the traditional notions of associativity and precedence of operators. (Unary minus causes a problem).

Using Operator-Precedence Relations

- The intention of the precedence relations is to find the handle of a right-sentential form,

  $\lessdot$ with marking the left end,

  $=\cdot$ appearing in the interior of the handle, and

  $\cdot>$ marking the right hand.

- In our input string $\$a_1a_2...a_n\$$, we insert the precedence relation between the pairs of terminals

Using Operator - Precedence Relations

$E \rightarrow E+E \mid E-E \mid E*E \mid E/E \mid E\hat{}E \mid (E) \mid -E \mid id$

The partial operator-precedence table for this grammar

	id	+	*	$
id		·>	·>	·>
+	<·	·>	<·	·>
*	<·	·>	·>	·>
$	<·	<·	<·	

- Then the input string (id + id*id) with the precedence relations inserted will be:

$ <· id ·> + <· id ·> * <· id ·> $

To Find The Handles

1. Scan the string from left end until the first ·> is encountered.
2. Then scan backwards (to the left) over any =· until a <· is encountered.
3. The handle contains everything to left of the first ·> and to the right of the <· is encountered.

$ <· id ·> + <· id ·> * <· id ·> $	$E \rightarrow id$	$ id + id * id $
$ <· + <· id ·> * <· id ·> $	$E \rightarrow id$	$ E + id * id $
$ <· + <· * <· id ·> $	$E \rightarrow id$	$ E + E * id $
$ <· + <· * ·> $	$E \rightarrow E*E$	$ E + E * ·E $
$ <· + ·> $	$E \rightarrow E+E$	$ E + E $
$ $		$ E $

Akshaya, Govt. Engg College Trivandrum

Operator-Precedence Parsing Algorithm

The input string is w$, the initial stack is $ and a table holds precedence relations between certain terminals

```
set p to point to the first symbol of w$ ;
repeat forever
    if ( $ is on top of the stack and p points to $ ) then return
    else {
        let a be the topmost terminal symbol on the stack and let b be the symbol pointed to by p;
        if ( a <· b or a = · b ) then {  /* SHIFT */
            push b onto the stack;
            advance p to the next input symbol;
        }
        else if ( a ·> b ) then          /* REDUCE */
            repeat pop stack
            until ( the top of stack terminal is related by <· to the terminal most recently popped );
        else error();
    }
```

Operator-Precedence Parsing Algorithm -- Example

stack	***input***	***action***
$	id+id*id$	$ <· id shift
$id	+id*id$	id ·> + reduceE → id
$	+id*id$	shift
$+	id*id$	shift
$+id	*id$	id ·> * reduceE → id
$+	*id$	shift
$+*	id$	shift
$+*id	$id ·> $	reduceE → id
$+*	$* ·> $	reduceE → E*E
$+	$+ ·> $	reduceE → E+E
$	$	accept

	id	+	*	$
id		·>	·>	·>
+	<·	·>	<·	·>
*	<·	·>	·>	·>
$	<·	<·	<·	

How to Create Operator-Precedence Relations

- We use associativity and precedence relations among operators.

1. If operator θ_1 has higher precedence than operator θ_2,
   ➔ $\theta_1 \cdot> \theta_2$ and $\theta_2 <\cdot \theta_1$
2. If operator θ_1 and operator θ_2 have equal precedence,
   they are left-associative ➔ $\theta_1 \cdot> \theta_2$ and $\theta_2 \cdot> \theta_1$
   they are right-associative ➔ $\theta_1 <\cdot \theta_2$ and $\theta_2 <\cdot \theta_1$
3. For all operators θ, $\theta <\cdot id$, $id \cdot> \theta$, $\theta <\cdot ($, $(<\cdot \theta, \theta \cdot>),) \cdot> \theta$, $\theta \cdot> \$$, and $\$ <\cdot \theta$
4. Also, let

$(=\cdot)$	$\$ <\cdot ($	$id \cdot>)$	$) \cdot> \$$
$(<\cdot ($	$\$ <\cdot id$	$id \cdot> \$$	$) \cdot>)$

Operator-Precedence Relations

	+	-	*	/	^	id	(	)	$
+	·>	·>	<·	<·	<·	<·	<·	·>	·>
-	·>	·>	<·	<·	<·	<·	<·	·>	·>
*	·>	·>	·>	·>	<·	<·	<·	·>	·>
/	·>	·>	·>	·>	<·	<·	<·	·>	·>
^	·>	·>	·>	·>	<·	<·	<·	·>	·>
id	·>	·>	·>	·>	·>			·>	·>
(	<·	<·	<·	<·	<·	<·	<·	=·	
)	·>	·>	·>	·>	·>			·>	·>
$	<·	<·	<·	<·	<·	<·	<·		

Akshaya, Govt. Engg College Trivandrum

Handling Unary Minus

- Operator-Precedence parsing cannot handle the unary minus when we also have the binary minus in our grammar.
- The best approach to solve this problem, let the lexical analyzer handle this problem.
  - The lexical analyzer will return two different tokens for the unary minus and the binary minus.
  - The lexical analyzer will need a lookhead to distinguish the binary minus from the unary minus.
- Then, we make

$\theta \lessdot$ unary-minus for any operator

unary-minus $\gtrdot \theta$ if unary-minus has higher precedence than θ

unary-minus $\lessdot \theta$ if unary-minus has lower (or equal) precedence than θ

Precedence Functions

- Compilers using operator precedence parsers do not need to store the table of precedence relations.
- The table can be encoded by two precedence functions f and g that map terminal symbols to integers.
- For symbols a and b.

  f(a) < g(b) whenever a $\lessdot$ b

  f(a) = g(b) whenever a $\doteq$ b

  f(a) > g(b) whenever a $\gtrdot$ b

Akshaya, Govt. Engg College Trivandrum

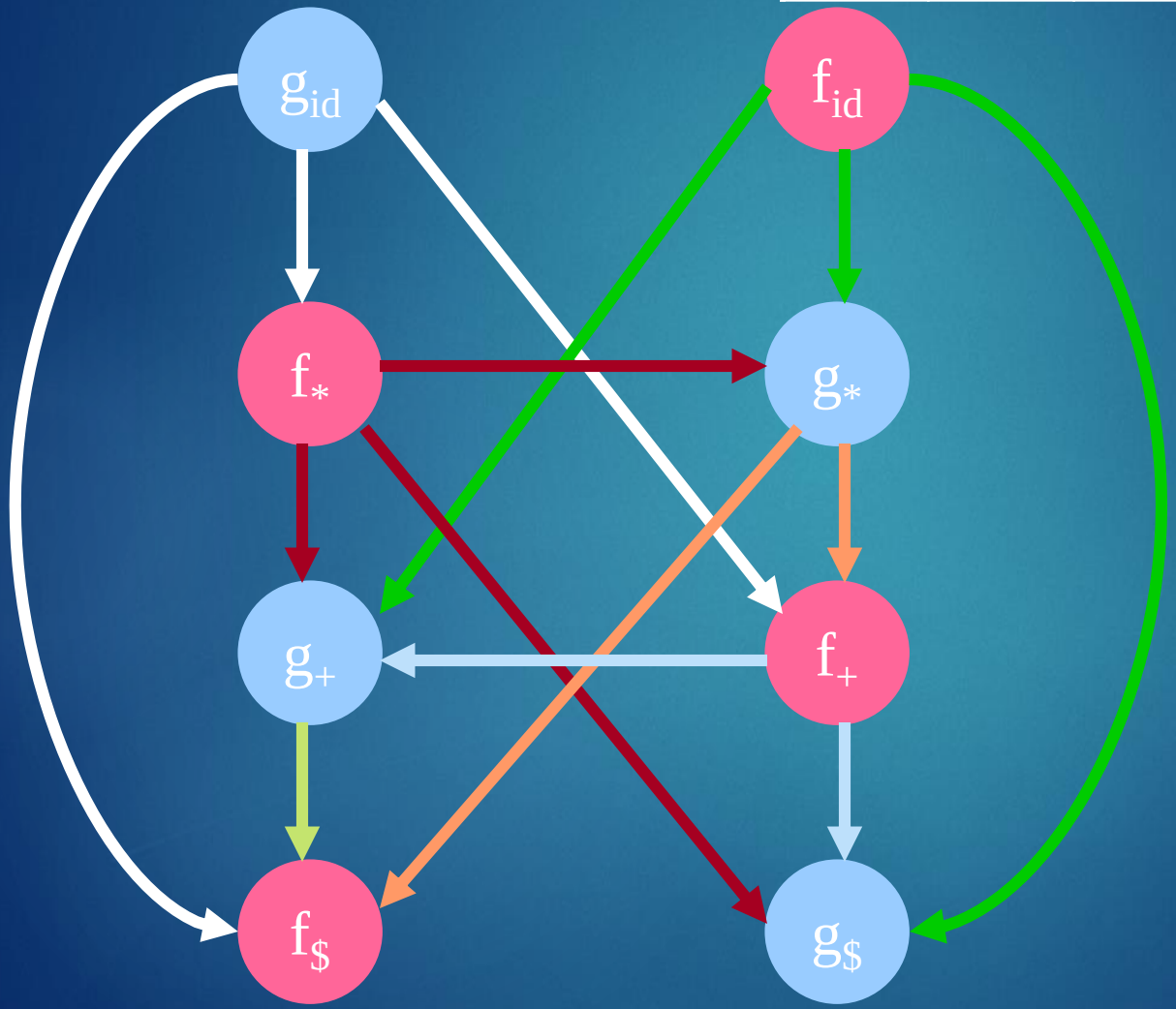
Constructing precedence functions

Method:

1. Create symbols $\mathbf{f_a}$ and $\mathbf{g_b}$ for each **a** that is a terminal or $.
2. Partition the created symbols into as many groups as possible, in such a way that if a =. b, then $\mathbf{f_a}$ and $\mathbf{g_b}$ are in the same group.
3. Create a directed graph whose nodes are the groups found in (2). For any a and b, if a <.b , place an edge from the group of $\mathbf{g_b}$ to the group of $\mathbf{f_a}$. Of a .> b, place an edge from the group of $\mathbf{f_a}$ to that of $\mathbf{g_b}$.
4. If the graph constructed has a cycle, then no precedence functions exist. If there are no cycle, let f(a) be the length of the longest path beginning at the group of $\mathbf{f_a}$; let g(a) be the length of the longest path beginning at the group of $\mathbf{g_a}$.

Example

	+	*	Id	$
f	2	4	4	0
g	1	3	5	0



Disadvantages of Operator Precedence Parsing

- **Disadvantages**:
  - It cannot handle the unary minus (the lexical analyzer should handle the unary minus).
  - Small class of grammars.
  - Difficult to decide which language is recognized by the grammar.

- **Advantages**:
  - simple
  - powerful enough for expressions in programming languages

Error Recovery in Operator-Precedence Parsing

Error Cases:

1. No relation holds between the terminal on the top of stack and the next input symbol.
2. A handle is found (reduction step), but there is no production with this handle as a right side

Error Recovery:

1. Each empty entry is filled with a pointer to an error routine.
2. Decides the popped handle "looks like" which right hand side. And tries to recover from that situation.

Akshaya, Govt. Engg College Trivandrum

Handling Shift/Reduce Errors

When consulting the precedence matrix to decide whether to shift or reduce, we may find that no relation holds between the top stack and the first input symbol.

To recover, we must modify (insert/change)

1. Stack or
2. Input or
3. Both.

We must be careful that we don't get into an infinite loop.

Example

	id	(	)	$
id	e3	e3	·>	·>
(	<·	<·	=·	e4
)	e3	e3	·>	·>
$	<·	<·	e2	e1

e1: Called when : whole expression is missing

insert **id** onto the input

issue diagnostic: ‘missing operand’

e2: Called when : expression begins with a right parenthesis

delete) from the input

issue diagnostic: ‘unbalanced right parenthesis’

Akshaya, Govt. Engg College Trivandrum

Example

	id	(	)	$
id	e3	e3	·>	·>
(	<·	<·	=.	e4
)	e3	e3	·>	·>
$	<·	<·	e2	e1

e3: Called when : id or) is followed by id or (

insert + onto the input

issue diagnostic: ‘missing operator’

e4: Called when : expression ends with a left parenthesis

pop (from the stack

issue diagnostic: ‘missing right parenthesis’

Akshaya, Govt. Engg College Trivandrum