# Report
# Topic: Operator Precedence Parsing(3rd Sep)

- Shashank Rajput (11CS10042)

## Precedence Relations

Bottom-up parsers for a large class of context-free grammars can be easily developed using *operator grammars*.

*Operator Grammars* have the property that **no production right side is empty or has two adjacent non-terminals**.

Consider:

E-> E op E | id
op-> + | *

Not an operator grammar but:

E-> E + E | E * E | id

This parser relies on the following three precedence relations:

| Relation | Meaning |
|---|---|
| *a* <· *b* | *a* yields precedence to *b* |
| *a* =· *b* | *a* has the same precedence as *b* |
| *a* ·> *b* | *a* takes precedence over *b* |

| | **id** | + | * | $ |
|---|---|---|---|---|
| **id** | | ·> | ·> | ·> |
| + | <· | ·> | <· | ·> |
| * | <· | ·> | ·> | ·> |
| $ | <· | <· | <· | ·> |

Precedence Table

*Example*: The input string:

**id**$_1$ + **id**$_2$ * **id**$_3$

After inserting precedence relations becomes:

$ <· **id**$_1$ ·> + <· **id**$_2$ ·> * <· **id**$_3$ ·> $

## Basic Principle

Having precedence relations allows identifying handles as follows:

1. Scan the string from left until seeing ·> and put a pointer.
2. Scan backwards the string from right to left until seeing <·
3. Everything between the two relations <· and ·> forms the handle
4. Replace handle with the head of the production.

## Operator Precedence Parsing Algorithm

*Initialize*: Set *ip* to point to the first symbol of the input string *w*$
***Repeat***: Let *b* be the top stack symbol, a the input symbol pointed to by *ip*

```
if (a is $ and b is $)
    return
else
    if $a \cdot> b$ or $a =\cdot b$ then
        push a onto the stack
        advance ip to the next input symbol
else if $a <\cdot b$ then
        repeat
            c ← pop the stack
        until (c .> stack-top)
    else error
end
```

## Making Operator Precedence Relations

The operator precedence parsers usually do not store the precedence table with the relations; rather they are implemented in a special way.
Operator precedence parsers use **precedence functions** that map terminal symbols to integers, and so the precedence relations between the symbols are implemented by numerical comparison.

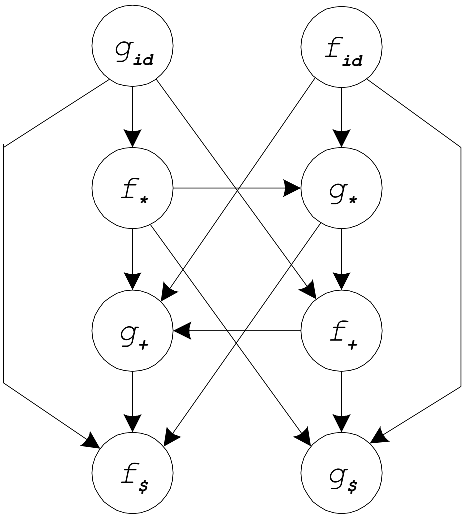
**Algorithm for Constructing Precedence Functions**

1. Create functions $f_a$ for each grammar terminal *a* and for the end of string symbol.
2. Partition the symbols in groups so that $f_a$ and $g_b$ are in the same group if $a =\cdot b$ (there can be symbols in the same group even if they are not connected by this relation).
3. Create a directed graph whose nodes are in the groups, next for each symbols a and b do: place an edge from the group of $g_b$ to the group of $f_a$ if $a <\cdot b$, otherwise if $a \cdot> b$ place an edge from the group of $f_a$ to that of $g_b$.
4. If the constructed graph has a cycle then no precedence functions exist. When there are no cycles collect the length of the longest paths from the groups of $f_a$ and $g_b$ respectively.

Example: consider the following table

| | **id** | + | * | $ |
|---|---|---|---|---|
| **id** | | ·> | ·> | ·> |
| + | <· | ·> | <· | ·> |
| * | <· | ·> | ·> | ·> |
| $ | <· | <· | <· | ·> |

Using the algorithm leads to the following graph:



From which we extract the following precedence functions:

| | **id** | + | * | $ |
|---|---|---|---|---|
| *f* | 4 | 2 | 4 | 0 |
| *g* | 5 | 1 | 3 | 0 |