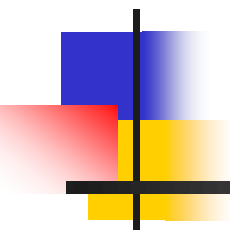


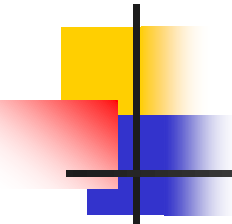
Programming Languages and Compilers (CS 421)



William Mansky

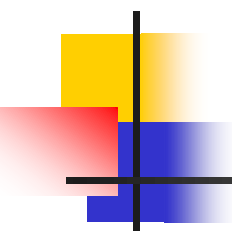
<http://courses.engr.illinois.edu/cs421/>

Based in part on slides by Mattox Beckman, as updated by Vikram Adve, Gul Agha, Elsa Gunter, and Dennis Griffith



Shift-Reduce (Bottom-Up) Parsing

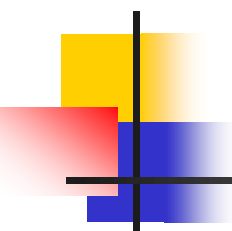
- Start at the bottom left and work your way up
- Keep track of string processed so far and remaining tokens
- At each step, either *shift* the next token onto the end of the processed string, or *reduce* the processed string by using a production in reverse
- Finished when entire string has been processed and reduced to start symbol
- More powerful (can handle more languages) than recursive descent



Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow$

$= \bullet (0 + 1) + 0$ shift



Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

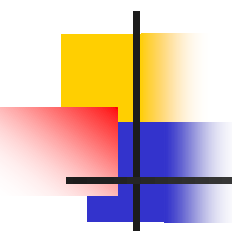
$\langle \text{Sum} \rangle \Rightarrow$

$$= (\text{●} 0 + 1) + 0$$

$$= \text{●} (0 + 1) + 0$$

shift

shift



Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow$

$$\Rightarrow (0 \bullet + 1) + 0$$

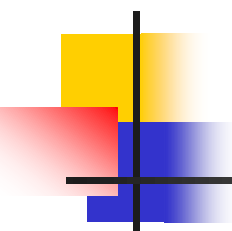
$$= (\bullet 0 + 1) + 0$$

$$= \bullet (0 + 1) + 0$$

reduce

shift

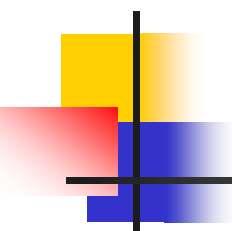
shift



Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow$

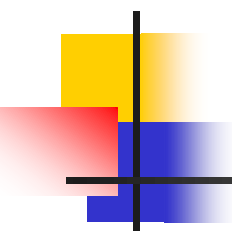
$=$	$(\langle \text{Sum} \rangle \bullet + 1) + 0$	shift
$\Rightarrow$	$(0 \bullet + 1) + 0$	reduce
$=$	$(\bullet 0 + 1) + 0$	shift
$=$	$\bullet (0 + 1) + 0$	shift



Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow$

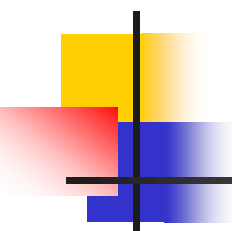
$= (\langle \text{Sum} \rangle + \bullet 1) + 0$	shift
$= (\langle \text{Sum} \rangle \bullet + 1) + 0$	shift
$\Rightarrow (0 \bullet + 1) + 0$	reduce
$= (\bullet 0 + 1) + 0$	shift
$= \bullet (0 + 1) + 0$	shift



Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow$

$\Rightarrow (\langle \text{Sum} \rangle + 1 \text{ } \bullet) + 0$	reduce
$= (\langle \text{Sum} \rangle + \text{ } \bullet 1) + 0$	shift
$= (\langle \text{Sum} \rangle \text{ } \bullet + 1) + 0$	shift
$\Rightarrow (0 \text{ } \bullet + 1) + 0$	reduce
$= (\text{ } \bullet 0 + 1) + 0$	shift
$= \text{ } \bullet (0 + 1) + 0$	shift



Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow$

$\Rightarrow (\langle \text{Sum} \rangle + \langle \text{Sum} \rangle \bullet) + 0$ reduce
 $\Rightarrow (\langle \text{Sum} \rangle + 1 \bullet) + 0$ reduce
 $= (\langle \text{Sum} \rangle + \bullet 1) + 0$ shift
 $= (\langle \text{Sum} \rangle \bullet + 1) + 0$ shift
 $\Rightarrow (0 \bullet + 1) + 0$ reduce
 $= (\bullet 0 + 1) + 0$ shift
 $= \bullet (0 + 1) + 0$ shift

Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow$

$= (\langle \text{Sum} \rangle \bullet) + 0$	shift
$\Rightarrow (\langle \text{Sum} \rangle + \langle \text{Sum} \rangle \bullet) + 0$	reduce
$\Rightarrow (\langle \text{Sum} \rangle + 1 \bullet) + 0$	reduce
$= (\langle \text{Sum} \rangle + \bullet 1) + 0$	shift
$= (\langle \text{Sum} \rangle \bullet + 1) + 0$	shift
$\Rightarrow (0 \bullet + 1) + 0$	reduce
$= (\bullet 0 + 1) + 0$	shift
$= \bullet (0 + 1) + 0$	shift

Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow$

$\Rightarrow (\langle \text{Sum} \rangle) \bullet + 0$	reduce
$= (\langle \text{Sum} \rangle \bullet) + 0$	shift
$\Rightarrow (\langle \text{Sum} \rangle + \langle \text{Sum} \rangle \bullet) + 0$	reduce
$\Rightarrow (\langle \text{Sum} \rangle + 1 \bullet) + 0$	reduce
$= (\langle \text{Sum} \rangle + \bullet 1) + 0$	shift
$= (\langle \text{Sum} \rangle \bullet + 1) + 0$	shift
$\Rightarrow (0 \bullet + 1) + 0$	reduce
$= (\bullet 0 + 1) + 0$	shift
$= \bullet (0 + 1) + 0$	shift

Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow$

$= \langle \text{Sum} \rangle \bullet + 0$	shift
$\Rightarrow (\langle \text{Sum} \rangle) \bullet + 0$	reduce
$= (\langle \text{Sum} \rangle \bullet) + 0$	shift
$\Rightarrow (\langle \text{Sum} \rangle + \langle \text{Sum} \rangle \bullet) + 0$	reduce
$\Rightarrow (\langle \text{Sum} \rangle + 1 \bullet) + 0$	reduce
$= (\langle \text{Sum} \rangle + \bullet 1) + 0$	shift
$= (\langle \text{Sum} \rangle \bullet + 1) + 0$	shift
$\Rightarrow (0 \bullet + 1) + 0$	reduce
$= (\bullet 0 + 1) + 0$	shift
$= \bullet (0 + 1) + 0$	shift

Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle \Rightarrow$

$= \langle \text{Sum} \rangle + \bullet 0$	shift
$= \langle \text{Sum} \rangle \bullet + 0$	shift
$\Rightarrow (\langle \text{Sum} \rangle) \bullet + 0$	reduce
$= (\langle \text{Sum} \rangle \bullet) + 0$	shift
$\Rightarrow (\langle \text{Sum} \rangle + \langle \text{Sum} \rangle \bullet) + 0$	reduce
$\Rightarrow (\langle \text{Sum} \rangle + 1 \bullet) + 0$	reduce
$= (\langle \text{Sum} \rangle + \bullet 1) + 0$	shift
$= (\langle \text{Sum} \rangle \bullet + 1) + 0$	shift
$\Rightarrow (0 \bullet + 1) + 0$	reduce
$= (\bullet 0 + 1) + 0$	shift
$= \bullet (0 + 1) + 0$	shift

Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

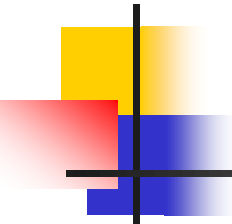
$\langle \text{Sum} \rangle$	$\Rightarrow$		
	$\Rightarrow$	$\langle \text{Sum} \rangle + 0 \bullet$	reduce
	$=$	$\langle \text{Sum} \rangle + \bullet 0$	shift
	$=$	$\langle \text{Sum} \rangle \bullet + 0$	shift
	$\Rightarrow$	$(\langle \text{Sum} \rangle) \bullet + 0$	reduce
	$=$	$(\langle \text{Sum} \rangle \bullet) + 0$	shift
	$\Rightarrow$	$(\langle \text{Sum} \rangle + \langle \text{Sum} \rangle \bullet) + 0$	reduce
	$\Rightarrow$	$(\langle \text{Sum} \rangle + 1 \bullet) + 0$	reduce
	$=$	$(\langle \text{Sum} \rangle + \bullet 1) + 0$	shift
	$=$	$(\langle \text{Sum} \rangle \bullet + 1) + 0$	shift
	$\Rightarrow$	$(0 \bullet + 1) + 0$	reduce
	$=$	$(\bullet 0 + 1) + 0$	shift
	$=$	$\bullet (0 + 1) + 0$	shift

Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

$\langle \text{Sum} \rangle$	$\Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle \bullet$	reduce
	$\Rightarrow \langle \text{Sum} \rangle + 0 \bullet$	reduce
	$= \langle \text{Sum} \rangle + \bullet 0$	shift
	$= \langle \text{Sum} \rangle \bullet + 0$	shift
	$\Rightarrow (\langle \text{Sum} \rangle) \bullet + 0$	reduce
	$= (\langle \text{Sum} \rangle \bullet) + 0$	shift
	$\Rightarrow (\langle \text{Sum} \rangle + \langle \text{Sum} \rangle \bullet) + 0$	reduce
	$\Rightarrow (\langle \text{Sum} \rangle + 1 \bullet) + 0$	reduce
	$= (\langle \text{Sum} \rangle + \bullet 1) + 0$	shift
	$= (\langle \text{Sum} \rangle \bullet + 1) + 0$	shift
	$\Rightarrow (0 \bullet + 1) + 0$	reduce
	$= (\bullet 0 + 1) + 0$	shift
	$= \bullet (0 + 1) + 0$	shift

Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

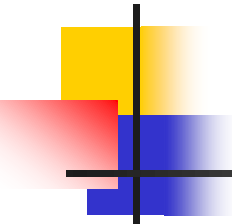
$\langle \text{Sum} \rangle \bullet$	$\Rightarrow \langle \text{Sum} \rangle + \langle \text{Sum} \rangle \bullet$	reduce
	$\Rightarrow \langle \text{Sum} \rangle + 0 \bullet$	reduce
	$= \langle \text{Sum} \rangle + \bullet 0$	shift
	$= \langle \text{Sum} \rangle \bullet + 0$	shift
	$\Rightarrow (\langle \text{Sum} \rangle) \bullet + 0$	reduce
	$= (\langle \text{Sum} \rangle \bullet) + 0$	shift
	$\Rightarrow (\langle \text{Sum} \rangle + \langle \text{Sum} \rangle \bullet) + 0$	reduce
	$\Rightarrow (\langle \text{Sum} \rangle + 1 \bullet) + 0$	reduce
	$= (\langle \text{Sum} \rangle + \bullet 1) + 0$	shift
	$= (\langle \text{Sum} \rangle \bullet + 1) + 0$	shift
	$\Rightarrow (0 \bullet + 1) + 0$	reduce
	$= (\bullet 0 + 1) + 0$	shift
	$= \bullet (0 + 1) + 0$	shift



Example

(0 + 1) + 0

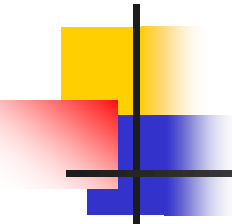




Example

(0 + 1) + 0

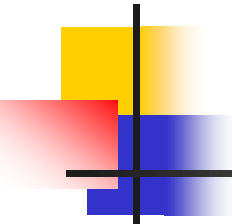




Example

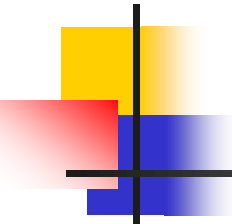
(0 + 1) + 0





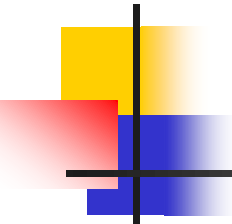
Example

$$\left(\begin{array}{c} \text{<Sum>} \\ | \\ 0 \end{array} \right) + 1 \quad) + 0$$

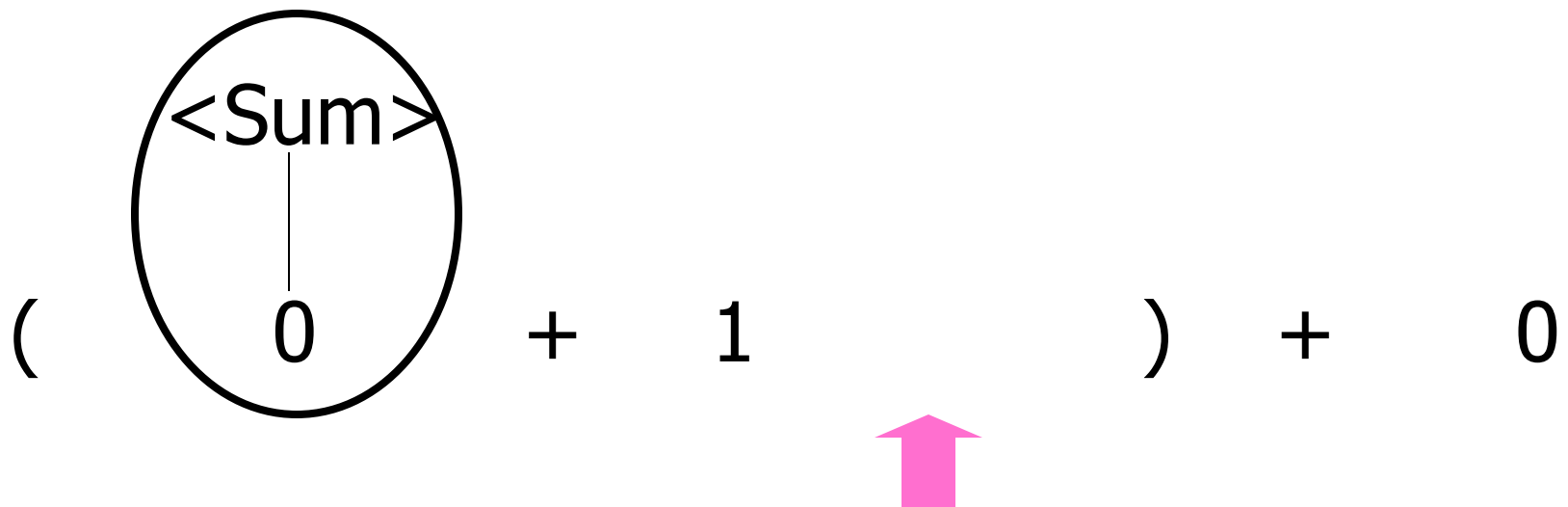



Example

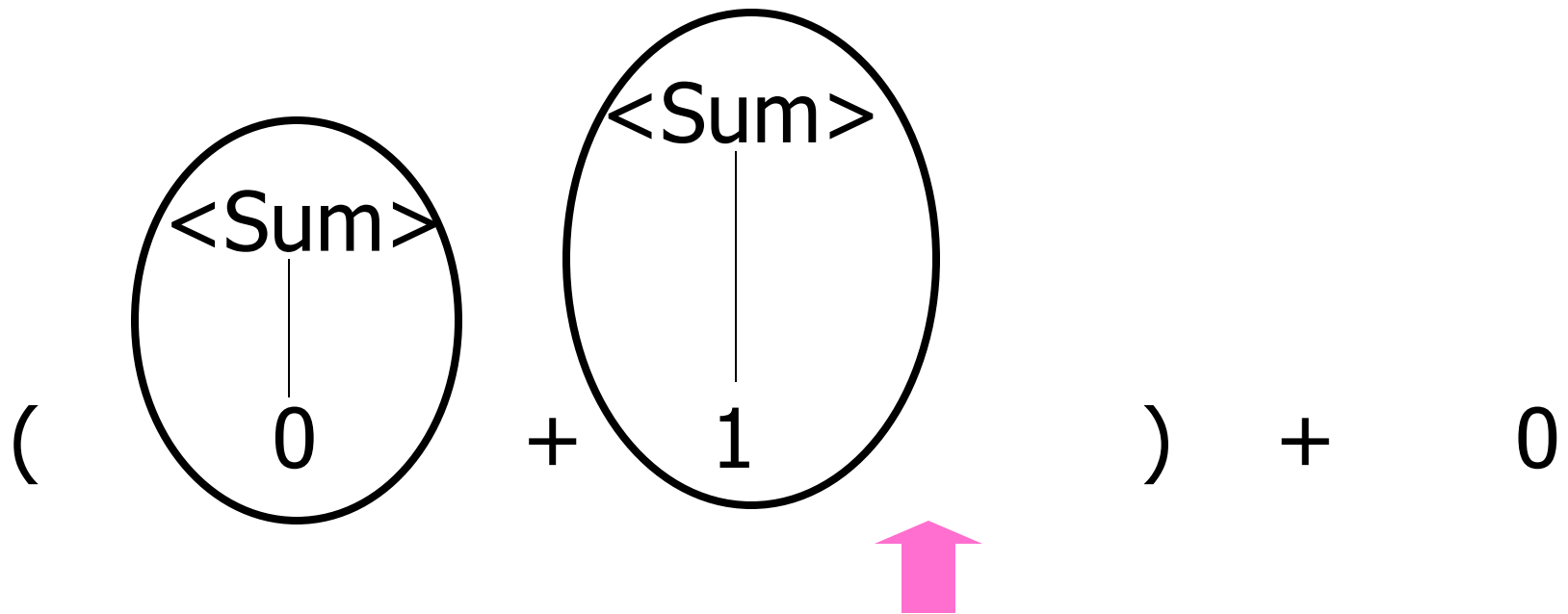
$$\left(\begin{array}{c} \text{<Sum>} \\ | \\ 0 \end{array} \right) + \underset{\uparrow}{1} \quad) + 0$$



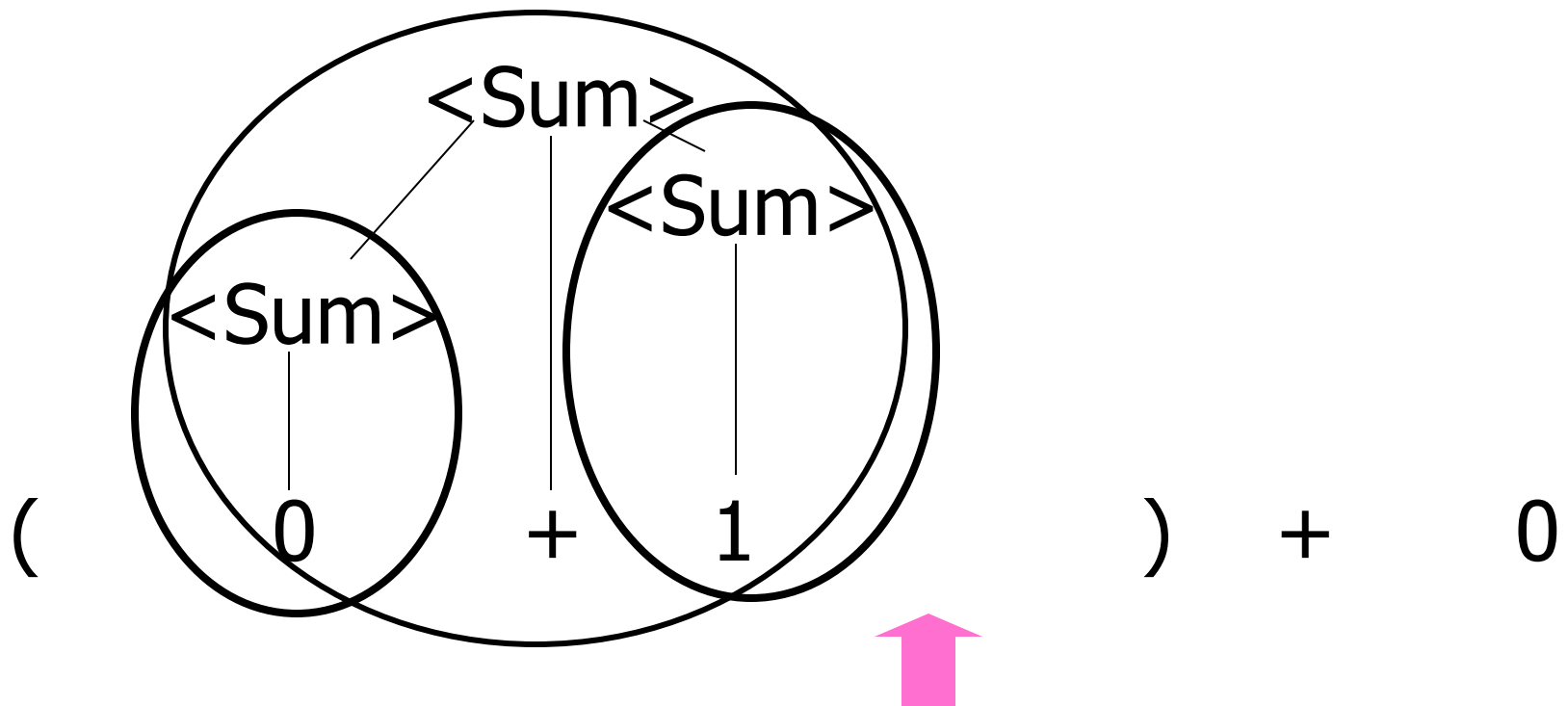
Example



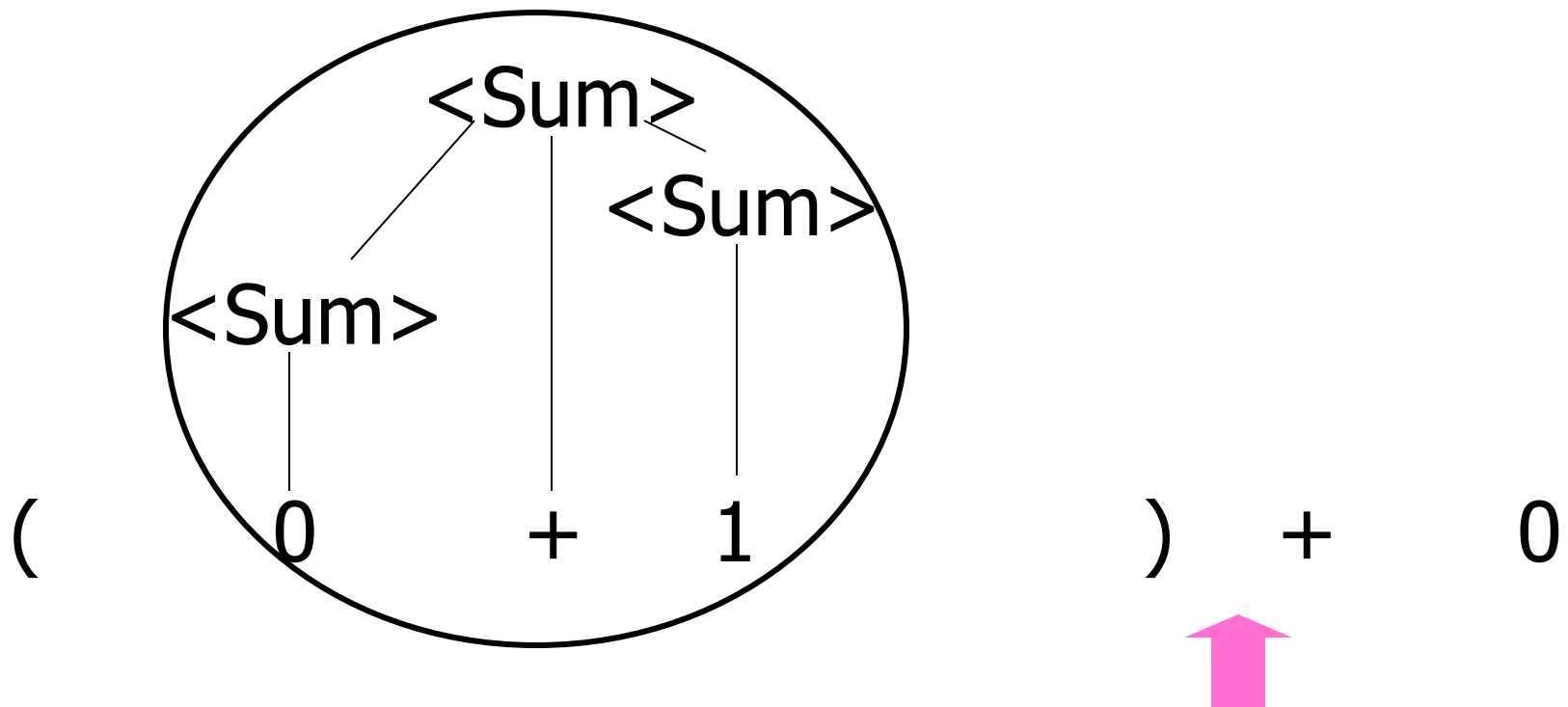
Example



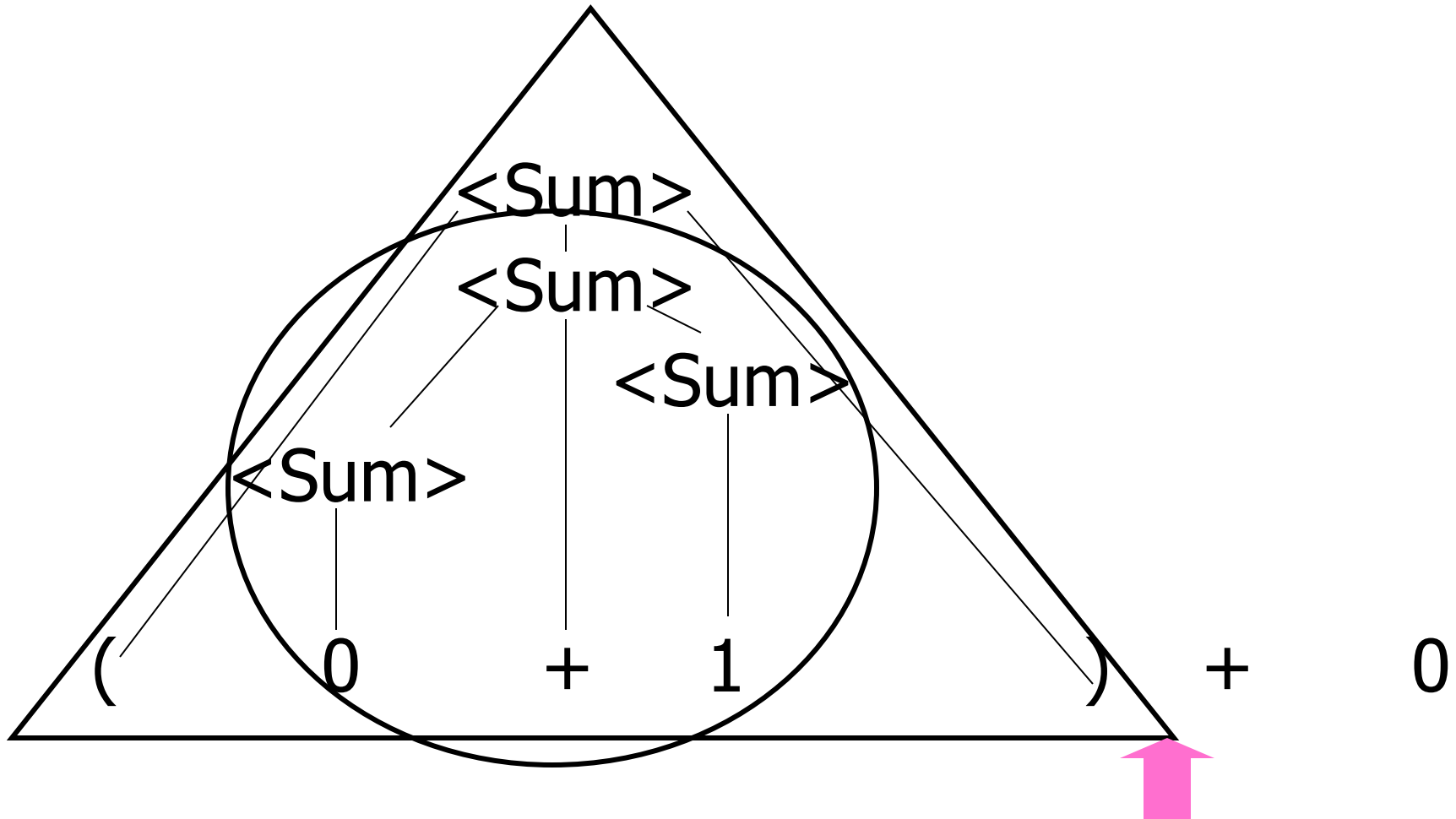
Example



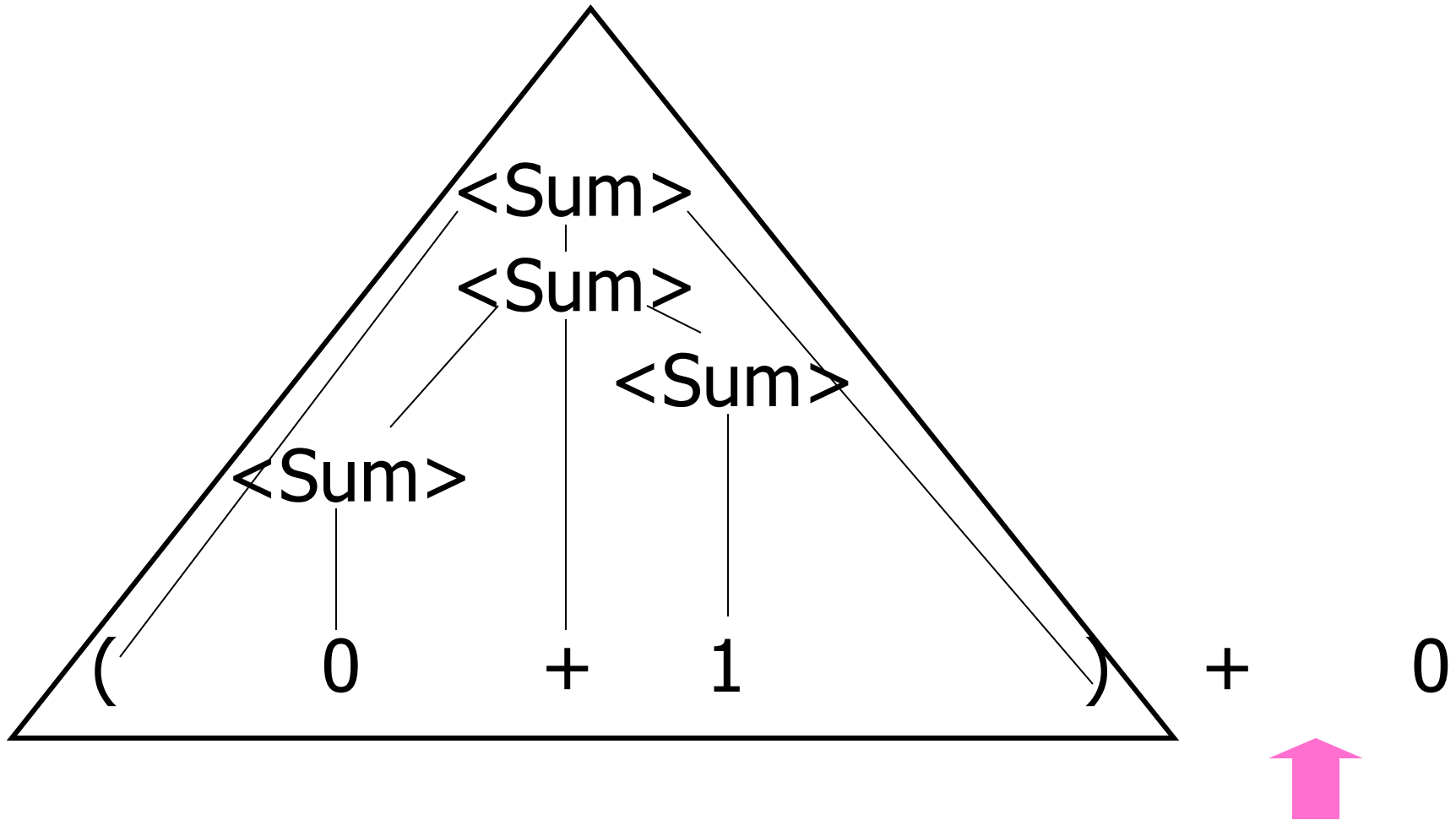
Example



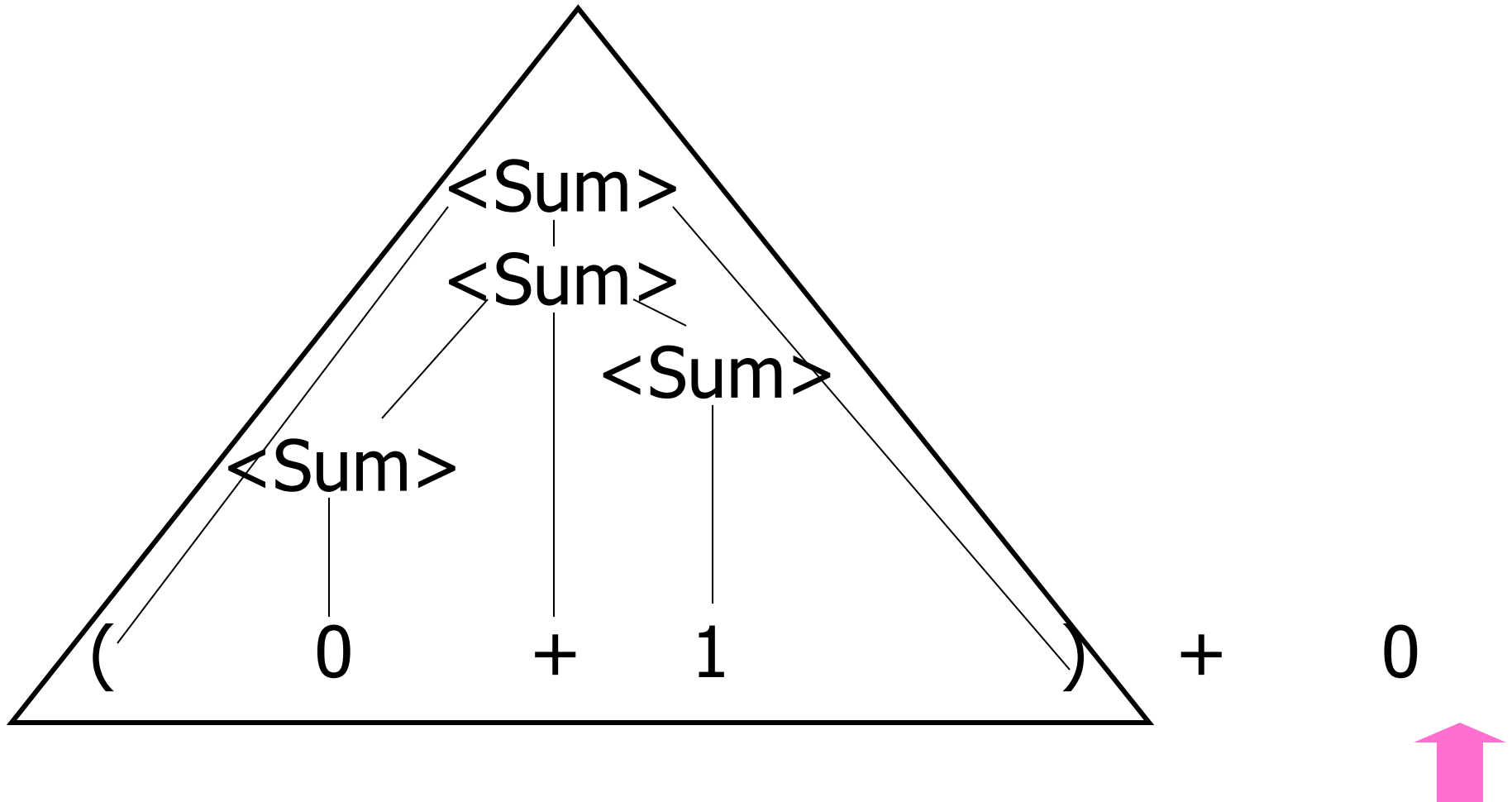
Example



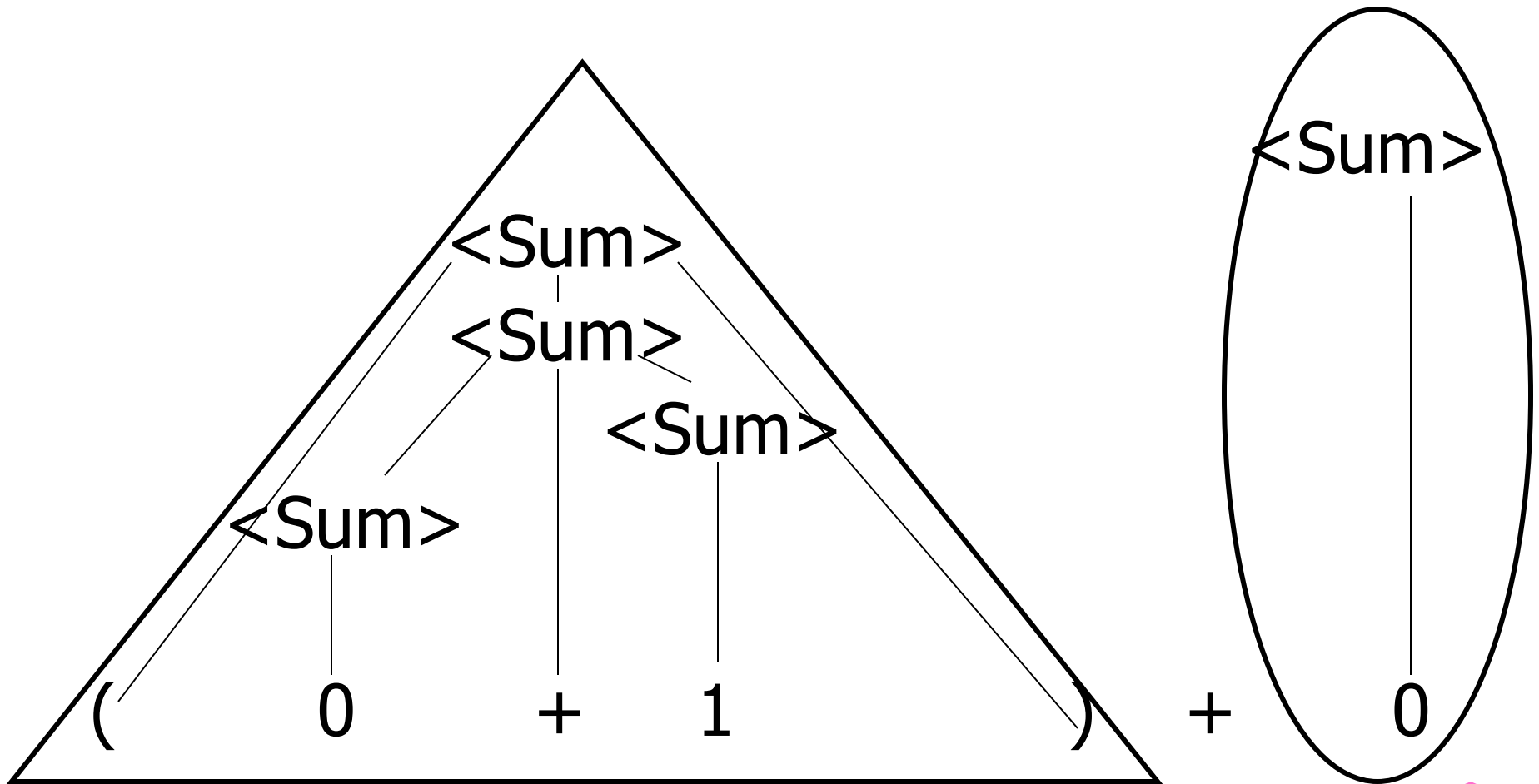
Example



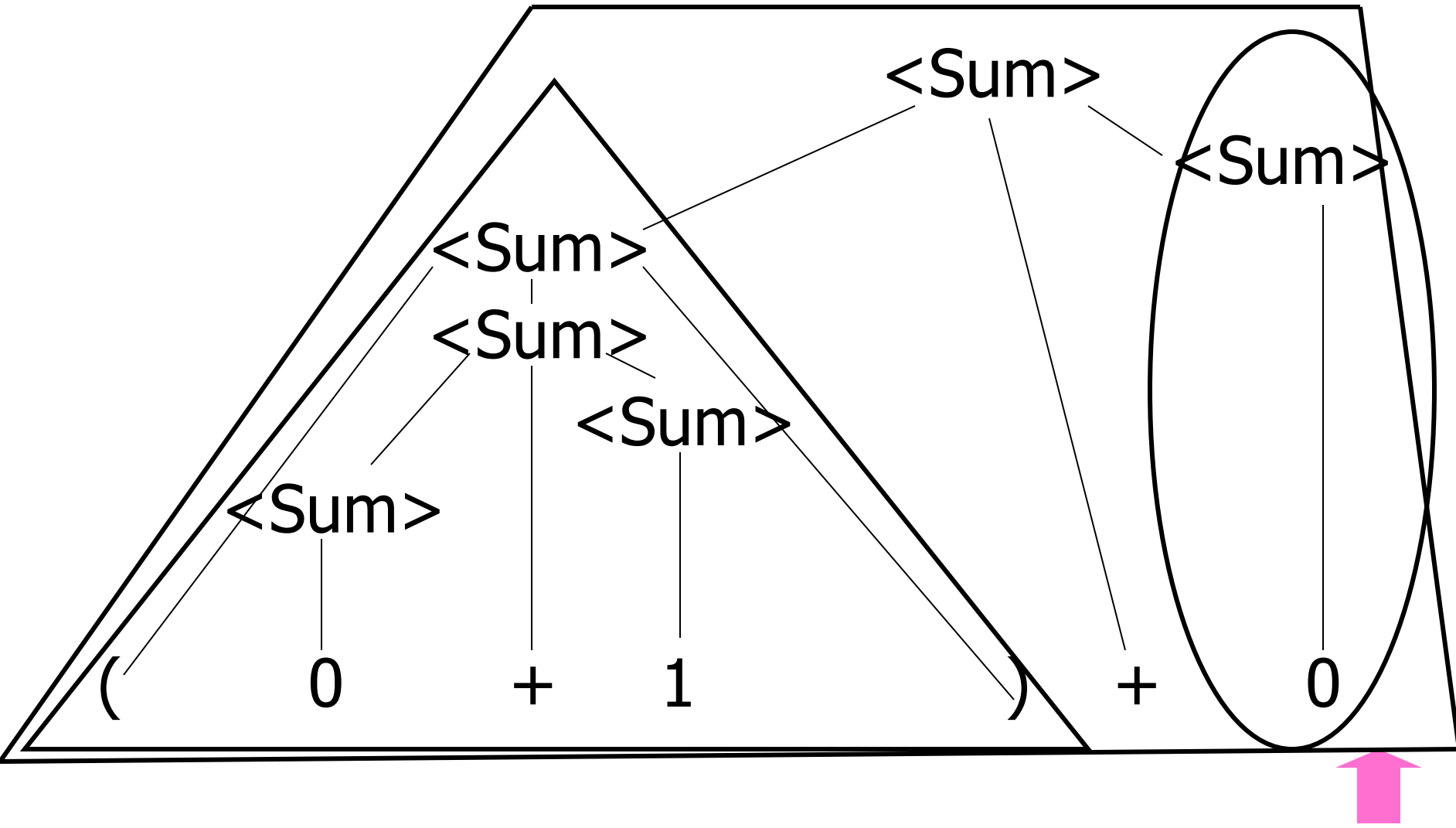
Example

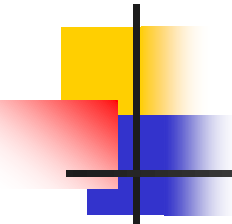


Example

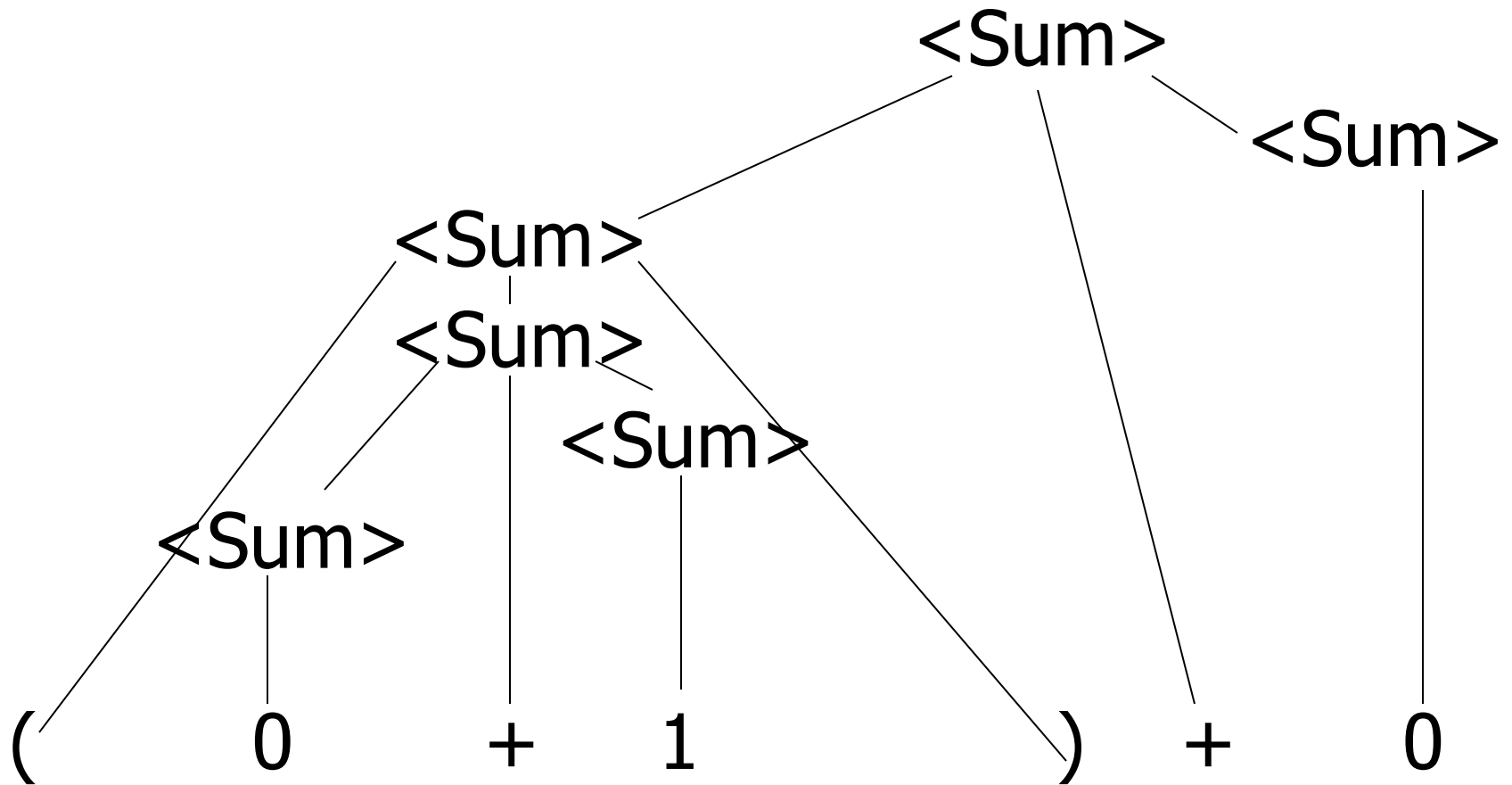


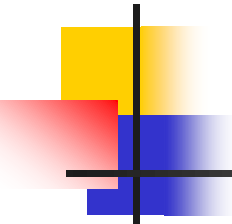
Example





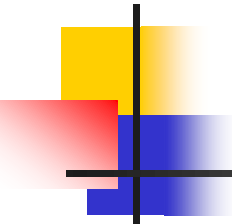
Example





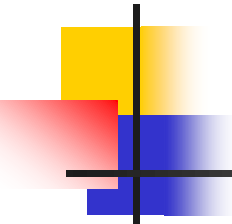
LR Parsing

- Shift-reduce reads tokens left to right (L)
- Creates a rightmost derivation (R), because we only reduce suffixes of the processed string
- Called LR parsing, vs. recursive descent's LL
- Can parse a wider class of languages than LL, and can be implemented efficiently



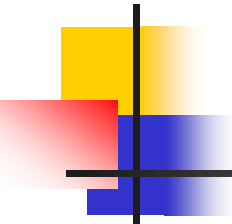
Implementing LR with Tables

- Build a pair of tables, Action and Goto, from the grammar
- LR works like state machine + stack where:
 - Action table tells whether to shift, reduce, accept, or fail in each state given next symbol
 - Goto table tells which state to add after reducing each non-terminal
 - Construction complicated, omitted here



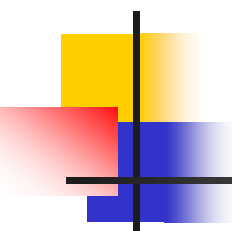
LR(k) Parsing Algorithm

- Based on push-down automata
- Uses states and transitions (as recorded in Action and Goto tables)
- Uses a stack containing states, terminals and non-terminals
- Look ahead k tokens when making decisions



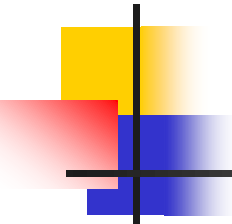
Shift-Reduce Conflicts

- **Problem:** can't decide whether the action for a state and input character should be **shift** or **reduce**
- Caused by ambiguity in grammar
- Usually caused by lack of associativity or precedence information in grammar



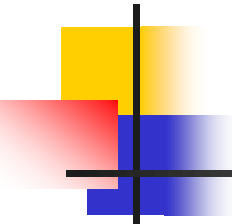
Example: $\langle \text{Sum} \rangle = 0 \mid 1 \mid (\langle \text{Sum} \rangle)$
 $\mid \langle \text{Sum} \rangle + \langle \text{Sum} \rangle$

● $0 + 1 + 0$ shift
-> $0 \text{ ●} + 1 + 0$ reduce
-> $\langle \text{Sum} \rangle \text{ ●} + 1 + 0$ shift
-> $\langle \text{Sum} \rangle + \text{ ●} 1 + 0$ shift
-> $\langle \text{Sum} \rangle + 1 \text{ ●} + 0$ reduce
-> $\langle \text{Sum} \rangle + \langle \text{Sum} \rangle \text{ ●} + 0$



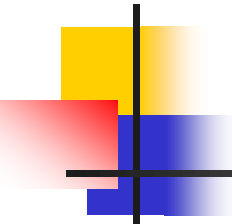
Example - cont

- **Problem:** shift or reduce?
- Choice between shift-shift-reduce-reduce and reduce-shift-shift-reduce
- Shift first: right-associative
- Reduce first: left-associative



Reduce-Reduce Conflicts

- **Problem:** can't decide between two different rules to reduce by
- Again caused by ambiguity in grammar
- **Symptom:** RHS of one production suffix of another
- Requires examining grammar and rewriting it
- Harder to solve than shift-reduce errors

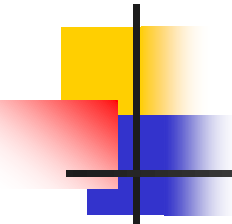


Example

■ $S ::= A \mid aB$ $A ::= abc$ $B ::= bc$

● abc	shift
a ● bc	shift
ab ● c	shift
abc ●	

■ Problem: reduce by $B ::= bc$ then by $S ::= aB$, or by $A ::= abc$ then $S ::= A$?



Using ocaml yacc

- Input grammar is put in file `<grammar>.mly`
- Execute

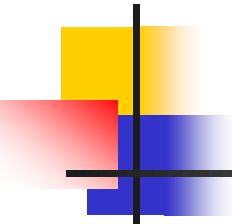
`ocaml yacc <grammar>.mly`

- Produces code for parser in

`<grammar>.ml`

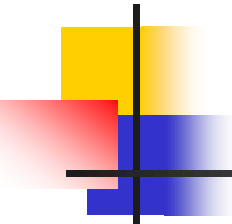
and interface (including type declaration for tokens) in

`<grammar>.mli`



Parser Code

- `<grammar>.ml` defines one parsing function per entry point
- Parsing function takes a lexing function (of type `lexing buffer -> token`) and a lexing buffer as arguments
- Returns value associated with corresponding entry point



ocamlyacc Input

- File format:

%{

<header>

%}

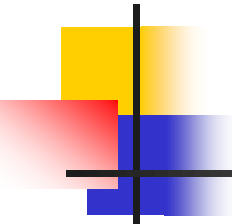
<declarations>

%%

<rules>

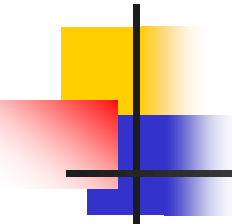
%%

<trailer>



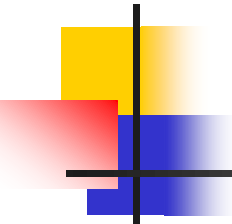
ocamlyacc *<header>*

- Contains arbitrary OCaml code
- Typically used to give types and functions needed for the semantic actions of rules and to give specialized error recovery
- May be omitted
- *<footer>* similar, can be used to call parser



ocamlyacc *<declarations>*

- **%token** *symbol ... symbol*
Declare given symbols as tokens/terminals
- **%token** *<type> symbol ... symbol*
Declare given symbols as token constructors, taking argument of type *<type>*
- **%start** *symbol ... symbol*
Declare given symbols as entry points; functions of same names will be generated in *<grammar>.ml*



ocamlyacc <declarations>

- **%type** <type> *symbol ... symbol*

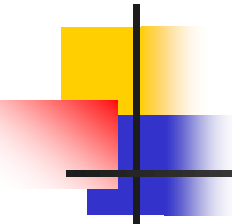
Specify type of values for given symbols;
mandatory for start symbols

- **%left** *symbol ... symbol*

- **%right** *symbol ... symbol*

- **%nonassoc** *symbol ... symbol*

Associate precedence and associativity to
given symbols. Same line, same precedence;
earlier line, lower precedence (broader
scope)



Ocamlyacc *<rules>*

- *nonterminal* :

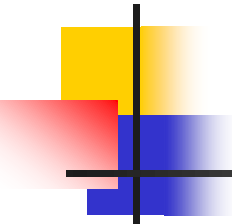
symbol ... symbol { semantic_action }

| ...

| *symbol ... symbol { semantic_action }*

;

- Semantic actions are arbitrary Ocaml expressions
- Must be of type declared (or inferred) for *nonterminal*
- Access values of symbols by position: \$1 for first symbol, \$2 for second, etc.



Example - Base types

```
(* File: expr.ml *)
```

```
type expr =
```

```
  Term_as_Expr of term
```

```
  | Plus_Expr of (term * expr)
```

```
  | Minus_Expr of (term * expr)
```

```
and term =
```

```
  Factor_as_Term of factor
```

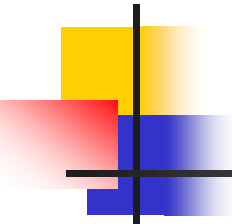
```
  | Mult_Term of (factor * term)
```

```
  | Div_Term of (factor * term)
```

```
and factor =
```

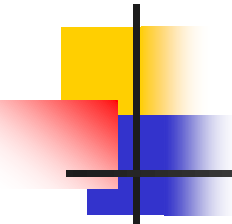
```
  Id_as_Factor of string
```

```
  | Parenthesized_Expr_as_Factor of expr
```



Example - Lexer (exprlex.mll)

```
{ (*open Exprparse*) }  
let numeric = ['0' - '9']  
let letter = ['a' - 'z' 'A' - 'Z']  
rule token = parse  
  | "+" {Plus_token}  
  | "-" {Minus_token}  
  | "*" {Times_token}  
  | "/" {Divide_token}  
  | "(" {Left_parenthesis}  
  | ")" {Right_parenthesis}  
  | letter (letter|numeric|"_")* as id {Id_token id}  
  | [' ' '\t' '\n'] {token lexbuf}  
  | eof {EOL}
```



Example - Parser (exprparse.mly)

```
%{ open Expr
```

```
%}
```

```
%token <string> Id_token
```

```
%token Left_parenthesis Right_parenthesis
```

```
%token Times_token Divide_token
```

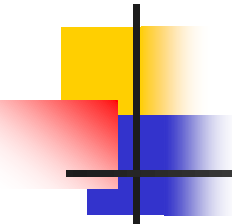
```
%token Plus_token Minus_token
```

```
%token EOL
```

```
%start main
```

```
%type <expr> main
```

```
%%
```



Example - Parser (exprparse.mly)

expr:

term

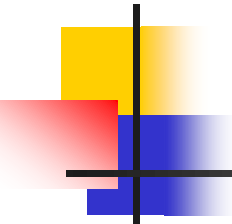
{ Term_as_Expr \$1 }

| term Plus_token expr

{ Plus_Expr (\$1, \$3) }

| term Minus_token expr

{ Minus_Expr (\$1, \$3) }



Example - Parser (exprparse.mly)

term:

factor

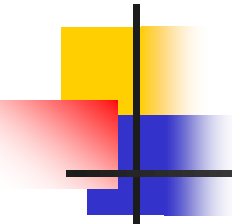
{ Factor_as_Term \$1 }

| factor Times_token term

{ Mult_Term (\$1, \$3) }

| factor Divide_token term

{ Div_Term (\$1, \$3) }



Example - Parser (exprparse.mly)

factor:

Id_token

{ Id_as_Factor \$1 }

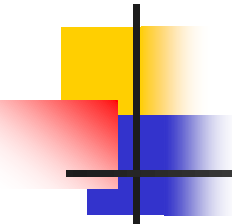
| Left_parenthesis expr Right_parenthesis

{Parenthesized_Expr_as_Factor \$2 }

main:

| expr EOL

{ \$1 }



Example - Using Parser

```
# #use "expr.ml";;
```

```
...
```

```
# #use "exprparse.ml";;
```

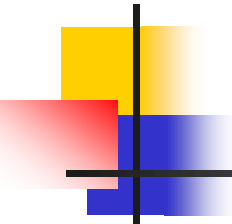
```
...
```

```
# #use "exprlex.ml";;
```

```
...
```

```
# let test s =
```

```
  let lexbuf = Lexing.from_string (s^"\n") in  
    main token lexbuf;;
```



Example - Using Parser

```
# test "a + b";;
```

```
- : expr =
```

```
Plus_Expr
```

```
(Factor_as_Term (Id_as_Factor "a"),  
 Term_as_Expr (Factor_as_Term  
   (Id_as_Factor "b"))))
```