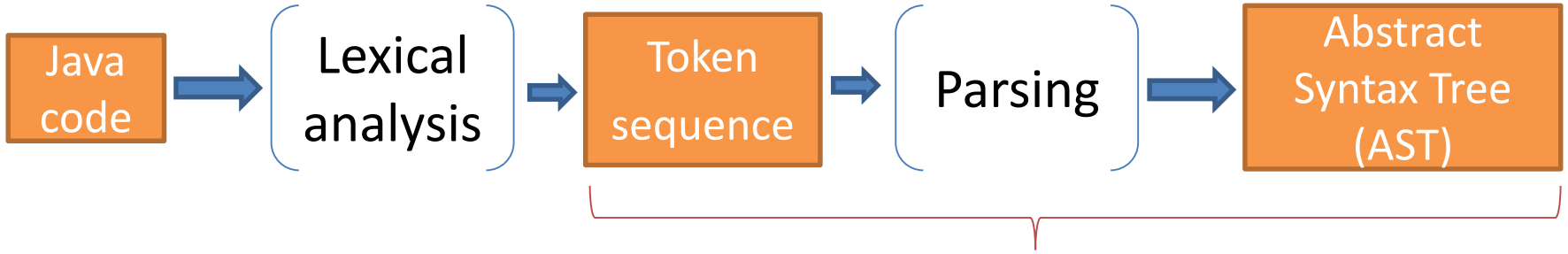
Compiler Construction
Winter 2020

Recitation 11:
Bottom-Up Parsing Conflicts

Yotam Feldman

Based on slides by the Technion compilers class' staff
and Guy Golan-Gueta

Lexing & Parsing



Shift/reduce conflicts

Reduce/reduce conflicts

Precedence

Associativity

Parsing

token stream

num(5)	+	(	num(7)	*	id(x)	)

Grammar:

$E \rightarrow \mathbf{id}$

$E \rightarrow \mathbf{num}$

$E \rightarrow E + E$

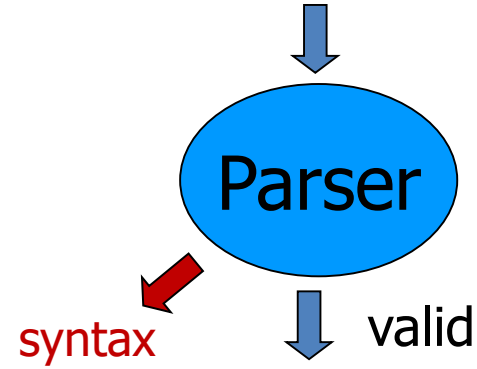
$E \rightarrow E - E$

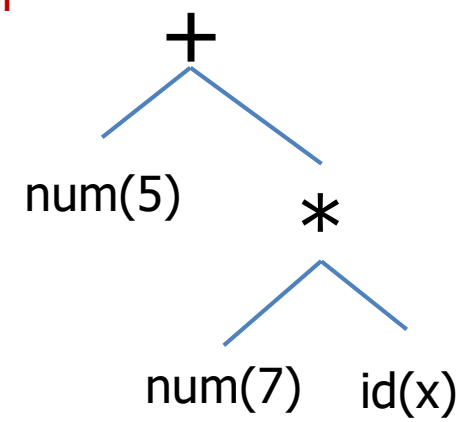
$E \rightarrow E * E$

$E \rightarrow E / E$

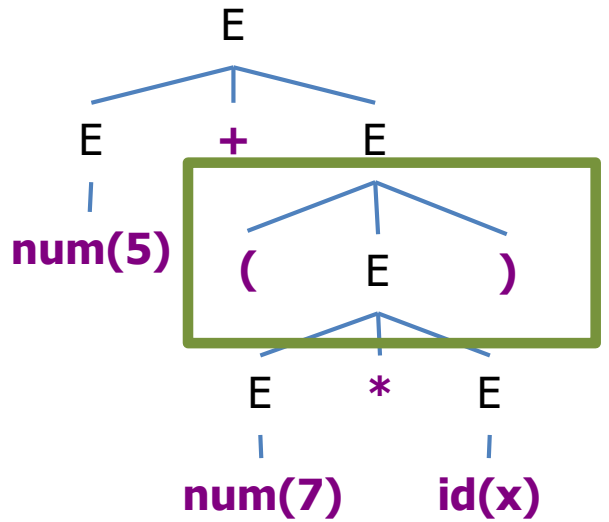
$E \rightarrow - E$

$E \rightarrow (E)$





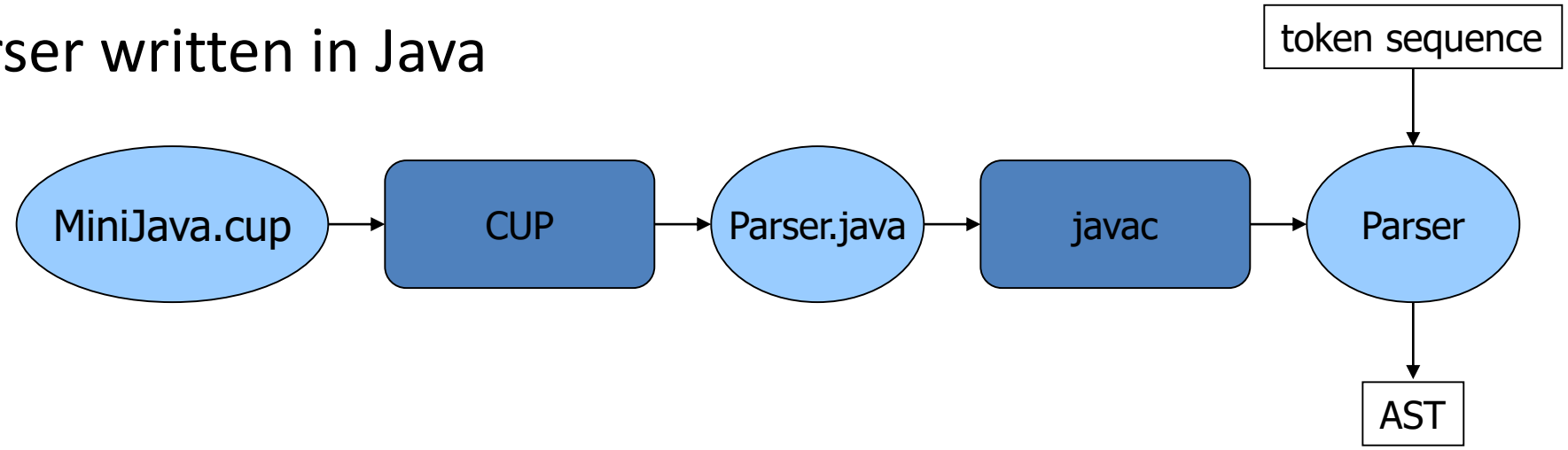
Abstract syntax tree



parse tree

CUP

- **C**onstructor of **U**seful **P**arsers
- Automatic LALR(1) parser generator
- Input
  - Parser specification file
- Output
  - Parser written in Java



Grammar in CUP

```
terminal int NUMBER;
terminal PLUS, MINUS, MULT, DIV;
terminal LPAREN, RPAREN;

non terminal expr;
```

```
start with expr;
```

```
expr ::= expr PLUS expr
       | expr MINUS expr
       | expr MULT expr
       | expr DIV expr
       | MINUS expr
       | LPAREN expr RPAREN
       | NUMBER;
```

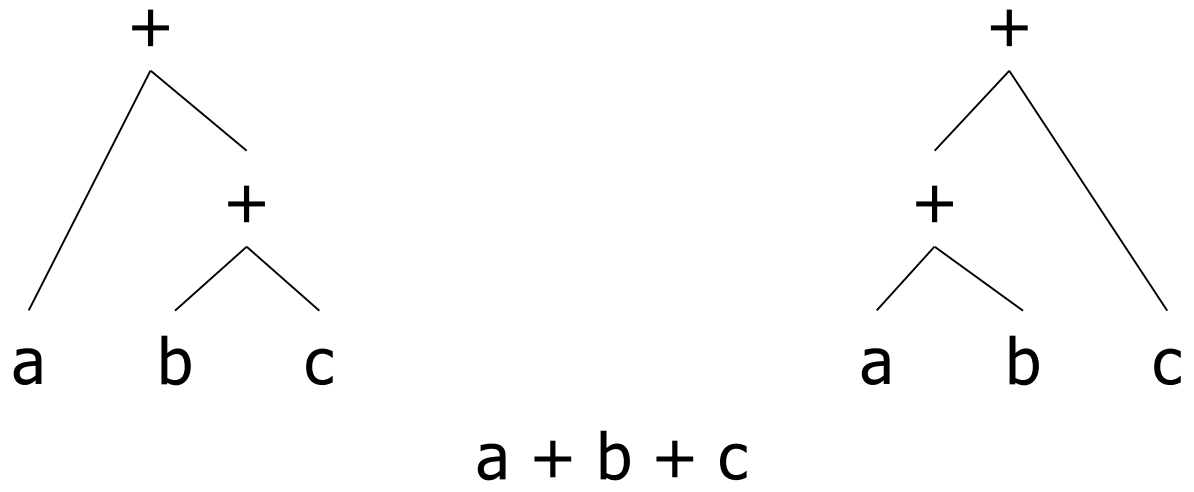
Lo and Behold

- See how it so majestically compiles!

- But…

Ambiguity

- A grammar is *ambiguous* if there exists a string that has two different rightmost derivations
- (A property of the grammar, not necessarily the language)
- Solutions:
  - Changing the grammar
  - Specifying precedence



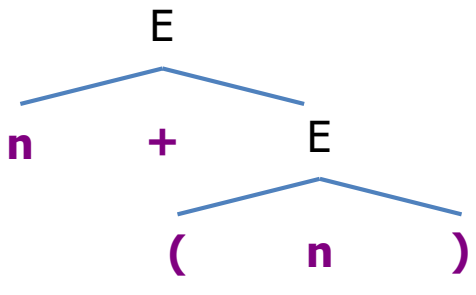
LR: Informal Recap

- Start with the final word, work way up to the starting symbol
  - Replacing a word with a non-terminal deriving it
- Read input from **left** to right
- Finding **rightmost** derivation, in the inverse order
- Stack of the derived word so far, automaton for deciding how to proceed
- Reduce: A prefix in the top of the stack matches the rhs of a derivation rule. Replace it with the lhs of that rule.
- Shift: A prefix in the top of the stack + lookahead is a prefix of the rhs of a derivation rule. Push the lookahead symbol to the stack

LR Parsing

input	look ahead	stack	operation
| 3 + (4)	n		shift
3 | + (4)	+	n	shift
3 + | (4)	(	n+	shift
3 + (| 4)	n	n+(	shift
3 + (4 |)	)	n+(n	shift
3 + (4) |		n+(n)	reduce
3 + (4) |		n+E	reduce
3 + (4) |		E	

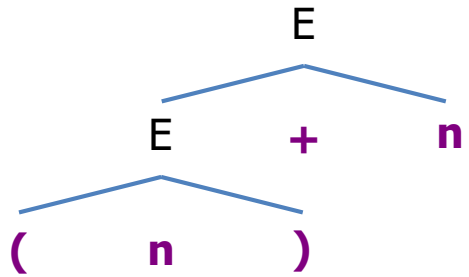
parse tree



LR Parsing

input	look ahead	stack	operation
| (4) + 3	(		shift
(| 4) + 3	n	(	shift
(4 |) + 3	)	(n	shift
(4) | + 3	+	(n)	**reduce**
(4) | + 3	+	E	shift
(4) + | 3	n	E+	shift
(4) + 3 |		E+n	reduce
(4) + 3 |		E	

parse tree



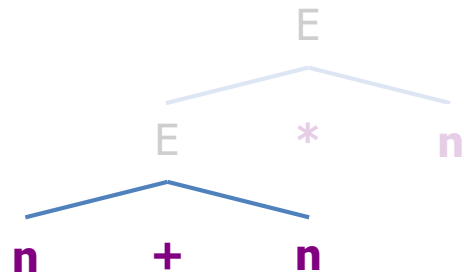
Shift/Reduce Conflicts

input	look ahead	stack	operation	parse tree
| 3 + 4 * 8	n		shift	
3 | + 4 * 8	+	n	shift	
3 + | 4 * 8	n	n+	shift	
3 + 4 | * 8	*	n+n	??	

Shift/Reduce Conflicts

input	look ahead	stack	operation
| 3 + 4 * 8	n		shift
3 | + 4 * 8	+	n	shift
3 + | 4 * 8	n	n+	shift
3 + 4 | * 8	*	n+n	**reduce**
3 + 4 | * 8	*	E	

parse tree



Top of the stack matches the rhs of a derivation rule
and the lookahead is possible in a derivation where this rule is applied

Shift/Reduce Conflicts

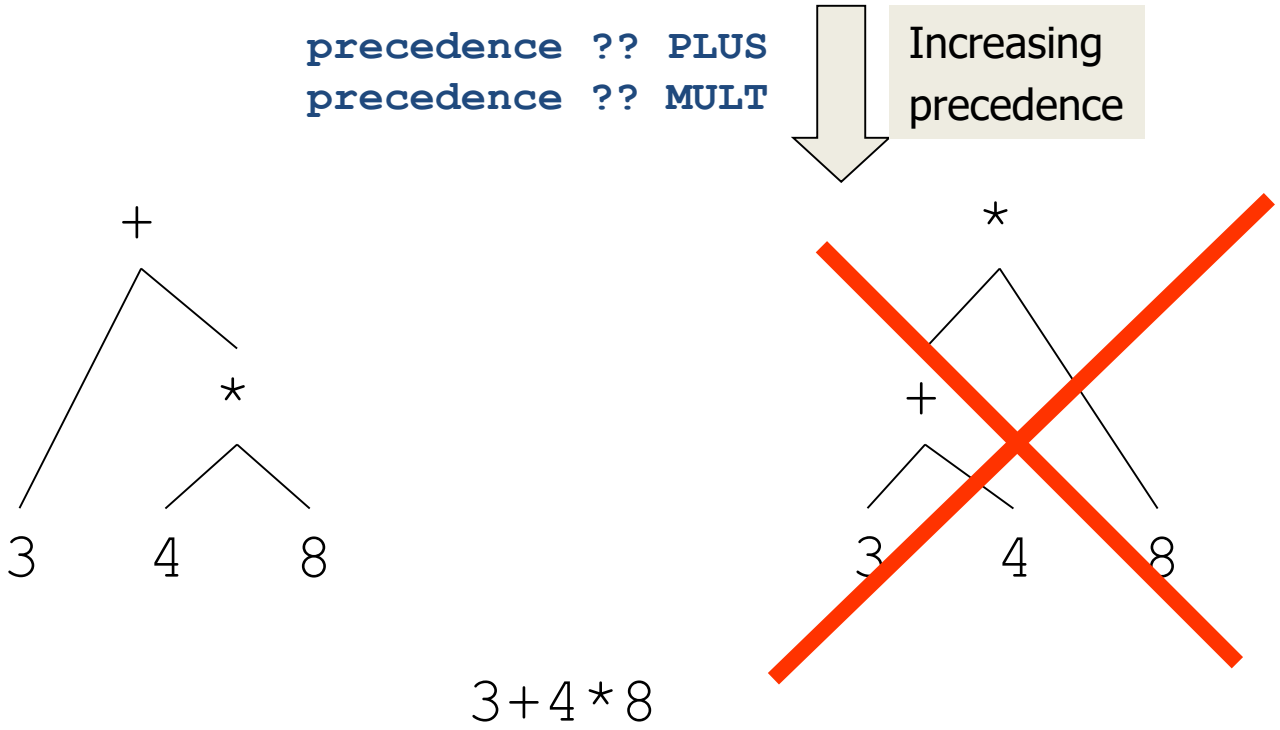
input	look ahead	stack	operation	parse tree
| 3 + 4 * 8	n		shift	
3 | + 4 * 8	+	n	shift	
3 + | 4 * 8	n	n+	shift	
3 + 4 | * 8	*	n+n	**shift**	
3 + 4 * | 8	n	n+n*		

Top of the stack + lookahead is a prefix of the rhs of a derivation rule

Shift/Reduce Conflicts

Resolving Ambiguity: Precedence

```
precedence ?? PLUS
precedence ?? MULT
```



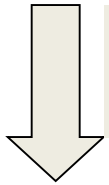
3+4*8

Precedence

```
terminal int NUMBER;
terminal PLUS, MINUS, MULT, DIV;
terminal LPAREN, RPAREN;

precedence PLUS, MINUS;
precedence DIV, MULT;

non terminal expr;
```



Increasing precedence

```
expr ::= expr PLUS expr
       | expr MINUS expr
       | expr MULT expr
       | expr DIV expr
       | MINUS expr
       | LPAREN expr RPAREN
       | NUMBER
```

Rule has precedence of PLUS

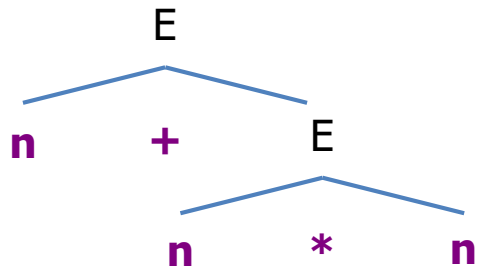
"When there is a shift/reduce conflict, the parser determines whether the terminal to be shifted has a higher precedence, or if the production to reduce by does. If the terminal has higher precedence, it is shifted"

Resolved via Precedence: Shift

input	look ahead	stack
| 3 + 4 * 8	n	
3 | + 4 * 8	+	n
3 + | 4 * 8	n	n+
3 + 4 | * 8	*	n+n
3 + 4 * | 8	n	n+n*
3 + 4 * 8 |		n+n*n
3 + 4 * 8 |		n+E
3 + 4 * 8 |		E

operation	parse tree
shift	
shift	
shift	
shift	
shift	
reduce	
reduce	

Reduce: precedence of +
Shift: precedence of *

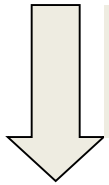


Precedence Beyond Conflict Resolution

```
terminal int NUMBER;
terminal PLUS, MINUS, MULT, DIV;
terminal LPAREN, RPAREN;

precedence ?? PLUS, MINUS;
precedence ?? DIV, MULT;

non terminal expr;
```



Increasing precedence

```
expr ::= expr PLUS expr
       | expr MINUS expr
       | expr MULT expr
       | expr DIV expr
       | MINUS expr
       | LPAREN expr RPAREN
       | NUMBER
```

What would happen with

- a * b

?

Precedence Beyond Conflict Resolution

```
terminal int NUMBER;
terminal PLUS, MINUS, MULT, DIV;
terminal UMINUS;
terminal LPAREN, RPAREN;

precedence ?? PLUS, MINUS;
precedence ?? DIV, MULT;
precedence ?? UMINUS;

non terminal    pr;
```

Increasing precedence

```
expr ::= expr PLUS expr
       | expr MINUS expr
       | expr MULT expr
       | expr DIV expr
       | MINUS expr %prec UMINUS
       | LPAREN expr RPAREN
       | NUMBER
```

Rule has precedence of UMINUS

(If not specified, of the last terminal in the production rule)

UMINUS never returned by lexer
(used only to define precedence)

Resolving Ambiguity: Associativity

"Associativity rules are also used to resolve shift/reduce conflicts, but only in the case of equal precedence."

```
precedence ?? PLUS
```

3+4+8

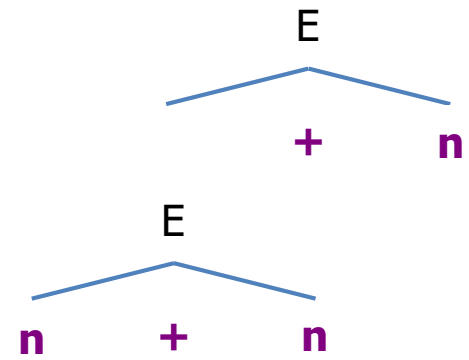
Shift/Reduce Conflicts: Associativity

input	look ahead	stack →	operation	parse tree
| 3 + 4 + 8	n		shift	
3 | + 4 + 8	+	n	shift	
3 + | 4 + 8	n	n+	shift	
3 + 4 | + 8	+	n+n	??	

precedence left PLUS

"If the associativity of the terminal that can be shifted is left, then a reduce is performed.

input	look ahead	stack	operation
| 3 + 4 + 8	n		shift
3 | + 4 + 8	+	n	shift
3 + | 4 + 8	n	n+	shift
3 + 4 | + 8	+	n+n	**reduce**
3 + 4 | + 8	+	E	shift
3 + 4 + | 8	n	E+	shift
3 + 4 + 8 |		E+E	reduce
3 + 4 + 8 |		E	

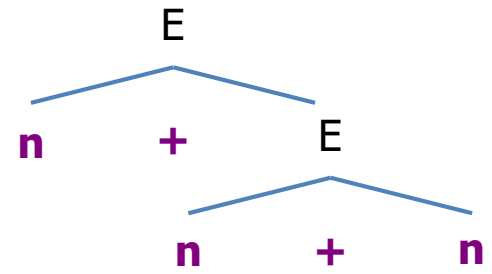


precedence right PLUS

"If the associativity of the terminal is right, it is shifted onto the stack. hence, the reductions will take place from right to left."

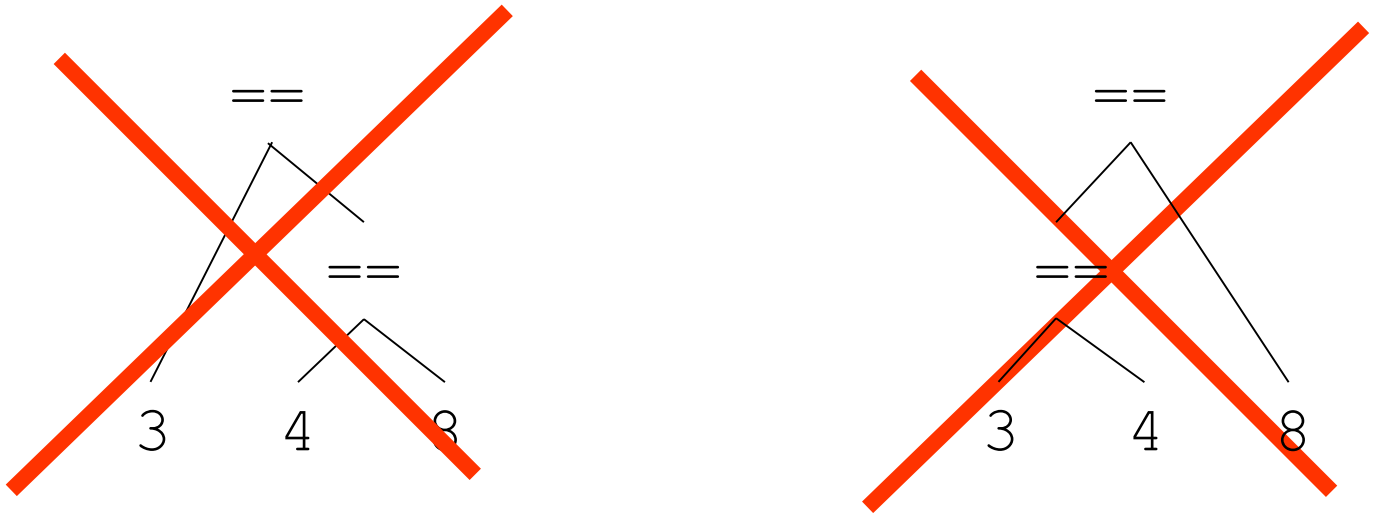
input	look ahead	stack	operation
| 3 + 4 + 8	n		shift
3 | + 4 + 8	+	n	shift
3 + | 4 + 8	n	n+	shift
3 + 4 | + 8	+	n+n	**shift**
3 + 4 + | 8	n	n+n+	shift
3 + 4 + 8 |		n+n+n	reduce
3 + 4 + 8 |		n+E	reduce
3 + 4 + 8 |		E	

parse tree



precedence nonassoc EQ

"If a terminal is declared as nonassoc, then two consecutive occurrences of equal precedence non-associative terminals generates an error."



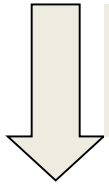
3==4==8

Precedence and Associativity

```
terminal int NUMBER;
terminal PLUS, MINUS, MULT, DIV;
terminal UMINUS;
terminal LPAREN, RPAREN;

precedence left PLUS, MINUS;
precedence left DIV, MULT;
precedence left UMINUS;

non terminal expr;
```



Increasing precedence

```
expr ::= expr PLUS expr
       | expr MINUS expr
       | expr MULT expr
       | expr DIV expr
       | MINUS expr %prec UMINUS
       | LPAREN expr RPAREN
       | NUMBER
```

Reduce/Reduce Conflicts

input	look ahead	stack	operation
|b a a	b		shift
b | a a	a	b	reduce **3 or 4??**

- 1. S → Aaa
- 2. S → Bab
- 3. A → b
- 4. B → b

Debugging CUP

- Getting internal representation
  - Command line options:
    - -dump_grammar
    - -dump_states
    - -dump_tables
    - -dump
  - Enabled in the demo's build.xml CUP ant task

What With All This Parsing Anyway?

- "Which Parsing Approach?", September 2020
  - "I spent over 20 years assuming that parsing is easy and that I didn't need to understand it properly in order to use it well. Alas, reality has been a cruel teacher, and in this post I want to share some of the lessons I've been forced to slowly learn and acknowledge."

Summary

- Bottom-up (LR) parsing
- Shift/reduce conflicts
- Precedence, associativity
- Reduce/reduce conflicts
- Ex 4