Lecture 15

Tuesday, October 12, 2021 1:50 PM

Resolving syntactic ambiguity

Int Expr ::= Int Lit

| Int Expr + Int Expr

"3 + 4 + 5" ⟶ (3 + 4) + 5

or

3 + (4 + 5)

Int Expr ::= Int Lit

| Int Expr + Int Expr

| Int Expr * Int Expr

"3 + 4 * 5" ⟶ (3 + 4) * 5

or

3 + (4 * 5)

In the case of + & −

Int Expr ::= Int Expr + Int Lit

IntExpr ::= IntExpr + IntLit

"3+(4+5)"

| IntExpr + (IntExpr)

Back to + & * :

Suggestion: Right side of * is always an integer literal

~~IntExpr ::= IntExpr * IntExpr~~

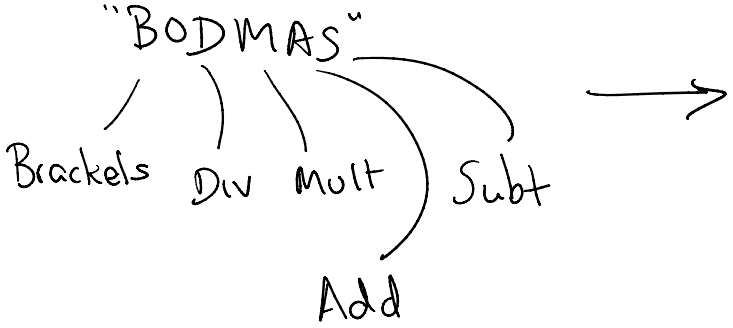
IntExpr ::= IntExpr * IntLit

What about "3*(4+5)" ?

| IntExpr * (IntExpr)

This suggests: Start with precedence

This suggests: Start with precedence rules.



Brackets · Mult · Add ⟹

$$Expr_0 ::= Expr_0 + Expr_1 \quad (+)$$
$$\mid Expr_1$$
$$Expr_1 ::= Expr_1 * Expr_2$$
$$\mid Expr_2$$
$$Expr_2 ::= Int\ Lit$$
$$\mid (\ Expr_0\ )$$

$3 * 4 + 5 * 6 + 8 * 3 * 4$

$3 * 4 + 5 * 6$: $Expr_0$

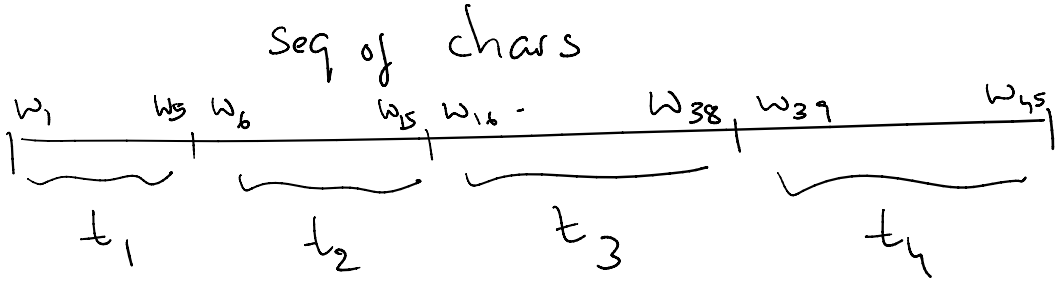
$8 * 3 * 4$: $Expr_1$

$8 * 3$: $Expr_1$ $\quad$ 4 : $Expr_2$

8 : $Expr_1$ $\quad$ 3 : $Expr_2$

Algorithmics

Lexical analyzer : seq of chars ⟶ seq of tokens



seq of tokens

Lexical analyzer : set of pattern matching rules to identify tokens

---

IntLit : ['0' – '9']+

An integer literal is a sequence of Ascii digits

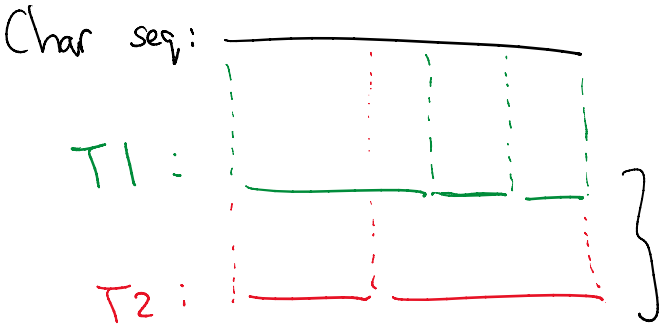
"9483 + 32" ⟶ IntLit 9483 ; PLUS ; IntLit 32

⟶ IntLit 948 ; IntLit 3 ; PLUS ; IntLit 32

⟶ IntLit 94 ; IntLit 83 ; PLUS ; IntLit 32

$\longrightarrow$ IntLit 94; IntLit 83; PLUS; IntLit 32

Ambiguity resolution using the Longest Match Principle.



Two alternative tokenizations

Swallow as many characters as possible to emit first token.

"Longest match principle"

Pattern matching in the lexical analyzer

- Set of rules
- Each rule checks if a sequence of characters is a token or not.

characters is a token or not.

"Regular expressions": Way to describe patterns over strings.

Examples: Word $:= \left(\left(\text{'a'} - \text{'z'} \right) \cup \left(\text{'A'} - \text{'Z'} \right) \right)^+$

(3) lower case letter — or — an (4) upper case letter

(2) each of which is a

Non-empty sequence of chars (1)

Email address : $\left[\left(\text{'a'} - \text{'z'} \right) \cup \left(\text{'A'} - \text{'Z'} \right) \cup \left(\text{'0'} - \text{'9'} \right) \right]^+ \cdot \text{'@'} \cdot \left[\text{'a'} - \text{'z'} \cup \text{'A'} - \text{'Z'} \cup \text{'0'} - \text{'9'} \right]^+ \cdot \text{'.'} \cdot [\ldots]^+$

Word · '@' · Word · '.' Word

Regular Expression, $r ::=$ Char Range (Digit, Uppercase letter, Vowel, ⋮)

$\mid r_1 \cdot r_2$

| $r_1 \cdot r_2$ \ vowel \ ⋮

String contains two parts
Left side matches r_1
Right side matches r_2

| $r_1 \cup r_2$

String matches r_1 or
String matches r_2

| r^*

Kleene *

String can be decomposed
(possibly empty) into a sequence.
Each piece matches r

| r^+

String can be decomposed into

a sequence of one or more pieces, each of which matches r.

$$r^+ = r \cdot r^* = r^* \cdot r$$