

## Context free grammar

$S ::= E \text{ EOF}$

$E ::= \lambda \text{ id } I . E \mid A R$

$I ::= \text{id } I \mid \varepsilon$

$A ::= \text{id} \mid ( E )$

$R ::= A R \mid \varepsilon$

**Non-Terminals:**  $S, E, I, A, R$

**Terminals:**  $\lambda, \text{id}, (, ), \text{EOF}$

# LL(1) Parsing Algorithm

```
var stack = List(startSymbol)
while (stack.nonEmpty && hasNextToken()) {
  val token = peekToken()
  val symbol = stack.head
  stack = stack.tail

  if (isTerminal(symbol)) {
    if (token == symbol) skipToken()
    else return false
  }
  else {
    chooseRule(symbol, token) match {
      case None => return false
      case Some(rule) => stack = rule ++ stack
    }
  }
}
if (stack.nonEmpty || hasNextToken()) {
  return false
}
return true
```

$S ::= E \text{ EOF}$

$E ::= \lambda \text{ id } I . E \mid A R$

$I ::= \text{id } I \mid \varepsilon$

$A ::= \text{id} \mid ( E )$

$R ::= A R \mid \varepsilon$

**Input:**  $\lambda \text{ id } . \text{id EOF}$

$S$

$E \text{ EOF}$

$\lambda \text{ id } I . E \text{ EOF}$

$\lambda \text{ id } . E \text{ EOF}$

$\lambda \text{ id } . A R \text{ EOF}$

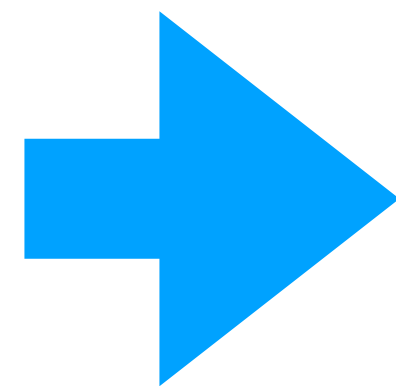
$\lambda \text{ id } . \text{id } R \text{ EOF}$

$\lambda \text{ id } . \text{id EOF}$

## Deciding which rule to take

A rule is chosen if:

- The rule **can start** with the next token
- The rule is **nullable**, and is **followed** by something that can start with the next token



**NULLABLE, FIRST, FOLLOW**

# NULLABLE

$\epsilon$  is nullable

Non-terminal  $A$  is nullable if there is a rule  
 $A ::= X_1 \dots X_n$  where  $X_i$  is nullable for all  $i$

$S ::= E \text{ EOF}$

$E ::= \lambda \text{ id } I . E \mid A R$

$I ::= \text{id } I \mid \epsilon$

$A ::= \text{id} \mid ( E )$

$R ::= A R \mid \epsilon$

# FIRST

For terminal  $x$ ,  $x \in \text{FIRST}(x)$

For non-terminal  $A$ ,  $x \in \text{FIRST}(A)$

if there exists a rule  $A ::= \textit{Pre B Post}$

for some (possibly empty) sequence of symbols  $\textit{Pre}$  and  $\textit{Post}$

where  $x \in \text{FIRST}(B)$  and

all symbols in  $\textit{Pre}$  are nullable

# FIRST

$S ::= E \text{ EOF}$

$E ::= \lambda \text{ id } I . E \mid A R$

$I ::= \text{id } I \mid \varepsilon$

$A ::= \text{id} \mid ( E )$

$R ::= A R \mid \varepsilon$

# FOLLOW

For non-terminal  $A$ ,  $x \in \mathbf{FOLLOW}(A)$   
if there exists a rule  $\mathbf{B} ::= \mathbf{Pre} \mathbf{A} \mathbf{Mid} \mathbf{C} \mathbf{Post}$   
for some (possibly empty) sequences of symbols  $Pre$ ,  $Mid$  and  $Post$   
where  $x \in \mathbf{FIRST}(C)$  and  
all symbols in  $\mathbf{Mid}$  are nullable

For non-terminal  $A$ ,  $x \in \mathbf{FOLLOW}(A)$   
if there exists a rule  $\mathbf{B} ::= \mathbf{Pre} \mathbf{A} \mathbf{Post}$   
for some (possibly empty) sequences of symbols  $Pre$  and  $Post$   
where  $x \in \mathbf{FOLLOW}(B)$  and  
all symbols in  $\mathbf{Post}$  are nullable



## FOLLOW

$S ::= E \text{ EOF}$

$E ::= \lambda \text{ id } I . E \mid A R$

$I ::= \text{id } I \mid \varepsilon$

$A ::= \text{id} \mid ( E )$

$R ::= A R \mid \varepsilon$

Computing NULLABLE, FIRST, FOLLOW

Gather **constraints**

Iteratively ensure constraints are satisfied  
until **fixpoint** is reached

# LL(1) Parsing Table

	$\lambda$	id	.	(	)	EOF
S						
E						
I						
A						
R						

Index of all rules of E which  
contain id in FIRST

Index of all rules of R  
which contain )  
in FIRST  
  
+  
  
If ) is in FOLLOW of R,  
index of all rules of R  
which are nullable

# LL(1) Parsing Table

	$\lambda$	id	.	(	)	EOF
S	1	1		1		
E	1	2		2		
I		1	2			
A		1		2		
R		1		1	2	2

## LL(1) Conflicts

Sometimes, LL(1) parsing table will contain multiple entries in the same cell.

Algorithm doesn't know which rule to apply simply looking at the next token.

## Fixing LL(1) Conflicts

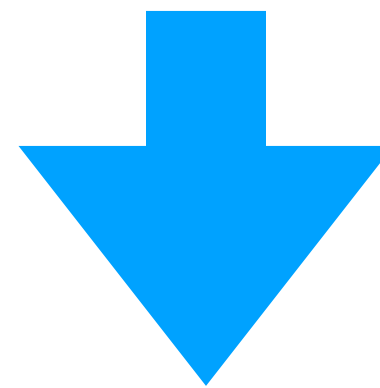


**Not always possible**, there are languages with context-free grammars but no LL(1) grammars.

# Fixing LL(1) Conflicts

## Left-factoring

$X ::= \mathbf{x} A \mid \mathbf{x} B$



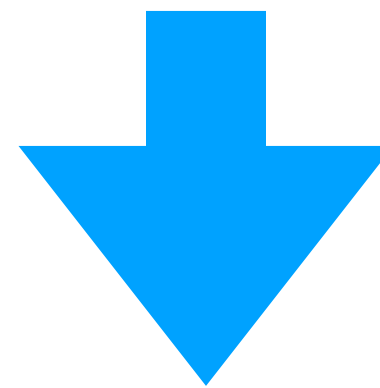
$X ::= \mathbf{x} Y$

$Y ::= A \mid B$

## Fixing LL(1) Conflicts

### Removing Left-recursion

$X ::= X + A \mid B$



$X ::= B R$

$R ::= + A R \mid \epsilon$



**Now, onto exercises !**