LL(1) Parsing

Exercise session 3

Context free grammar

```
S ::= E EOF
E ::= λ id I . E | A R
I ::= id I | ε
A ::= id | ( E )
R ::= A R | ε
```

Non-Terminals: `S, E, I, A, R`

Terminals: **`λ, id, (, ), 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 **EOF**

E ::= **λ id** I **.** E | A R

I ::= **id** I | ε

A ::= **id** | **(** E **)**

R ::= A R | ε

Input: **λ id . id EOF**

S

E **EOF**

λ id I **.** E **EOF**

λ id . E **EOF**

λ id . A R **EOF**

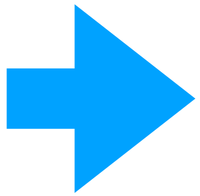
λ id . id R **EOF**

λ id . 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

ε is nullable

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

```
S ::= E EOF
E ::= λ id I . E | A R
I ::= id I | ε
A ::= id | ( E )
R ::= A R | ε
```

FIRST

For terminal x, x ∈ FIRST(x)

For non-terminal A, **x** ∈ **FIRST(A)**
if there exists a rule **A ::= *Pre* B *Post***
for some (possibly empty) sequence of symbols *Pre* and *Post*
where **x** ∈ **FIRST(B)** and
all symbols in ***Pre* are nullable**

FIRST

S ::= E **EOF**

E ::= **λ id** I **.** E | A R

I ::= **id** I | ε

A ::= **id** | **(** E **)**

R ::= A R | ε

FOLLOW

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

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

FOLLOW

S ::= E **EOF**

E ::= **λ id** I **.** E | A R

I ::= **id** I | ε

A ::= **id** | **(** E **)**

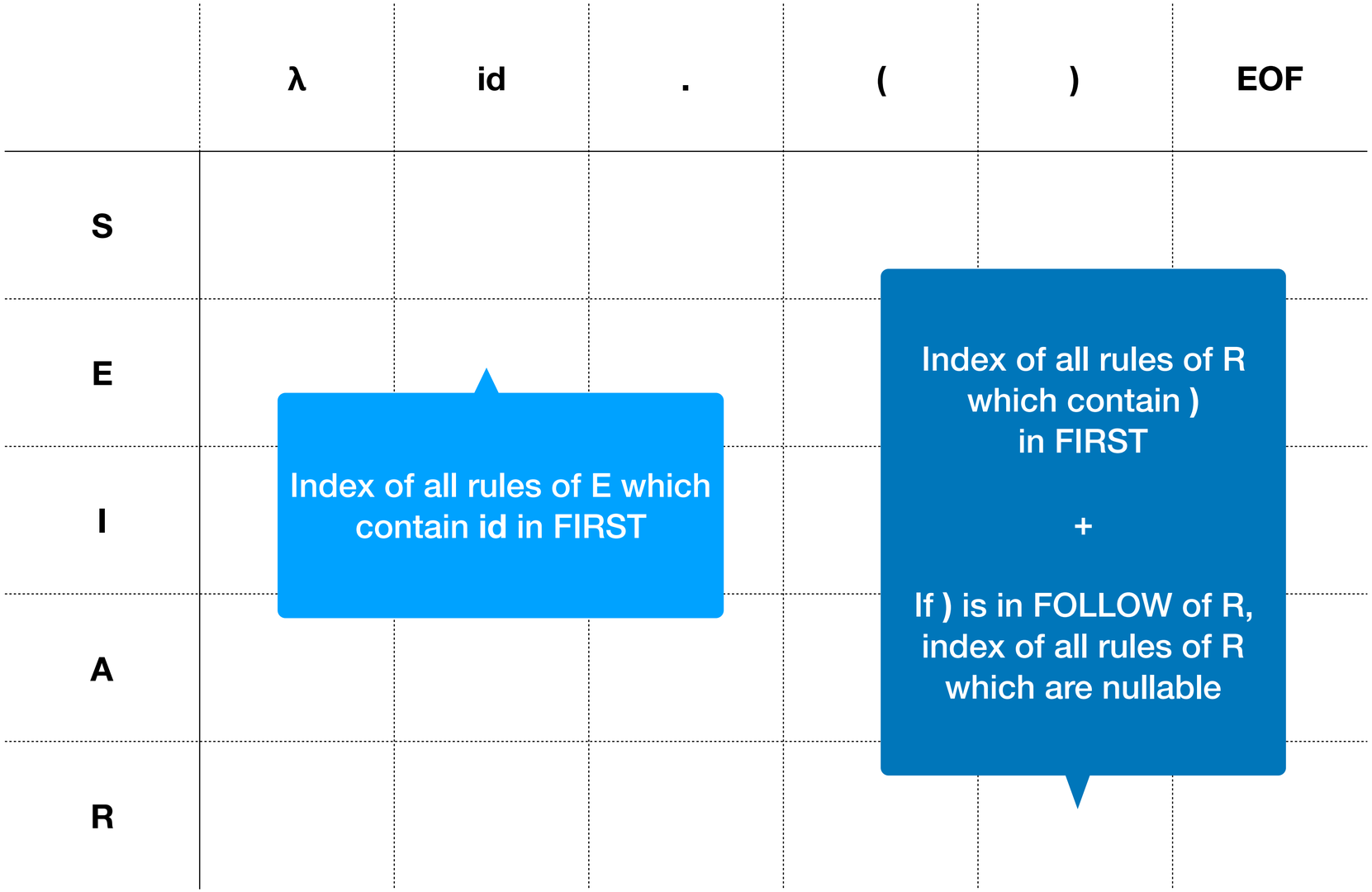
R ::= A R | ε

Computing NULLABLE, FIRST, FOLLOW

Gather **contraints**

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

LL(1) Parsing Table



LL(1) Parsing Table

	λ	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.

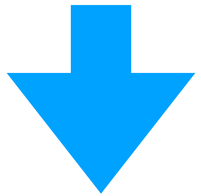
Fixing LL(1) Conflicts

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

Fixing LL(1) Conflicts

Left-factoring

X ::= **x** A | **x** B



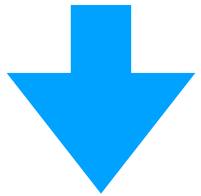
X ::= **x** Y

Y ::= A | B

Fixing LL(1) Conflicts

Removing Left-recursion

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



$$X ::= B\ R$$

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

Now, onto exercises !