

Concepts of Programming Languages – Recitation 1: Predictive Parsing

Oded Padon

`odedp@mail.tau.ac.il`

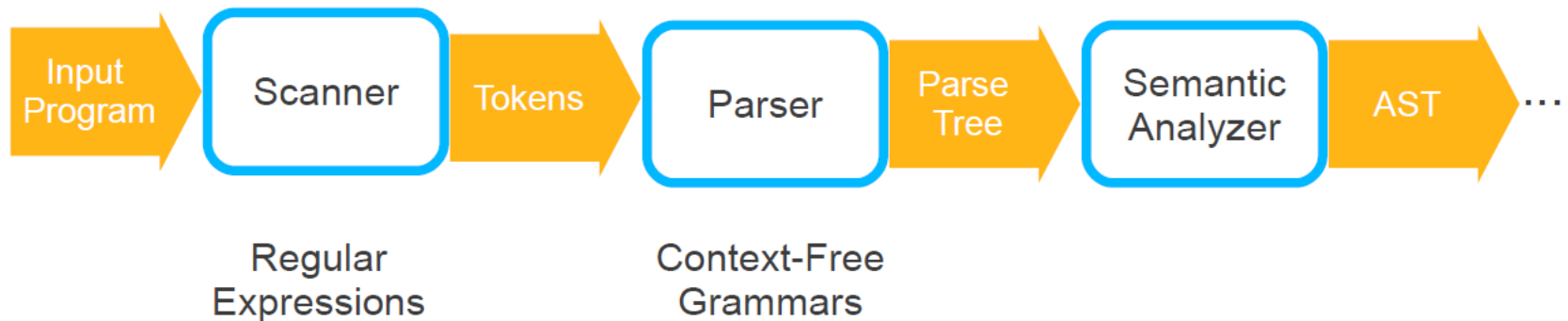
Reference:

Modern Compiler Implementation in Java by Andrew W. Appel – Ch. 3

Administrative

- Course website and forum (**not using moodle**):
 - <http://cs.tau.ac.il/~msagiv/courses/pl16.html>
 - <http://groups.google.com/forum/#!forum/taupl16>
- Recitation groups:
 - **Wednesday 15:10-16:00 Kaplun 118**
 - **New group, please register to this group if you can!**
 - Wednesday 16:10-17:00 Shenkar 220
 - Thursday 15:10-16:00 Shenkar 220
- Grade: 50% Exercises, 50% Exam
- Exercises: **all mandatory**, teams of 1 or 2 or 3 students
- First exercise will be published soon
- My reception hours: Wednesdays, 17:00-18:00
 - email before you come!

Role of Parsing



Context Free Grammars

- Terminals (tokens)
- Non-terminals
 - Start non-terminal
- Derivation rules (also called productions)
 $\langle \text{Non-Terminal} \rangle \rightarrow \text{Symbol Symbol} \dots \text{Symbol}$

Example Context Free Grammar

- Terminals (tokens)
- Non-terminals
 - Start non-terminal

1 $\langle S \rangle \rightarrow \langle S \rangle ; \langle S \rangle$

2 $\langle S \rangle \rightarrow \text{id} := \langle E \rangle$

3 $\langle S \rangle \rightarrow \text{print} (\langle L \rangle)$

4 $\langle E \rangle \rightarrow \text{id}$

5 $\langle E \rangle \rightarrow \text{num}$

6 $\langle E \rangle \rightarrow \langle E \rangle + \langle E \rangle$

7 $\langle E \rangle \rightarrow (\langle S \rangle, \langle E \rangle)$

8 $\langle L \rangle \rightarrow \langle E \rangle$

9 $\langle L \rangle \rightarrow \langle L \rangle, \langle E \rangle$

Example Parse Tree

<S>

<S> ; <S>

<S> ; id := E

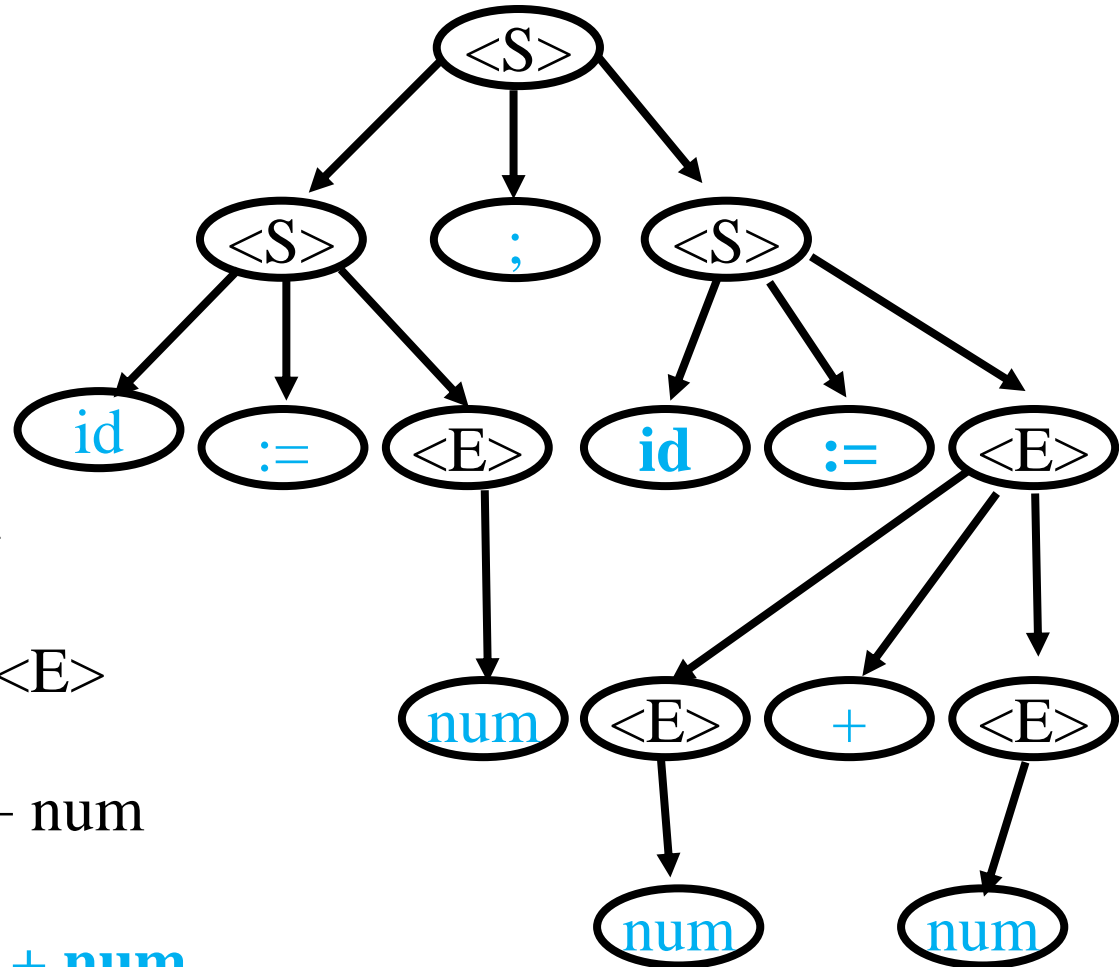
id := <E> ; id := <E>

id := num ; id := <E>

id := num ; id := <E> + <E>

id := num ; id := <E> + num

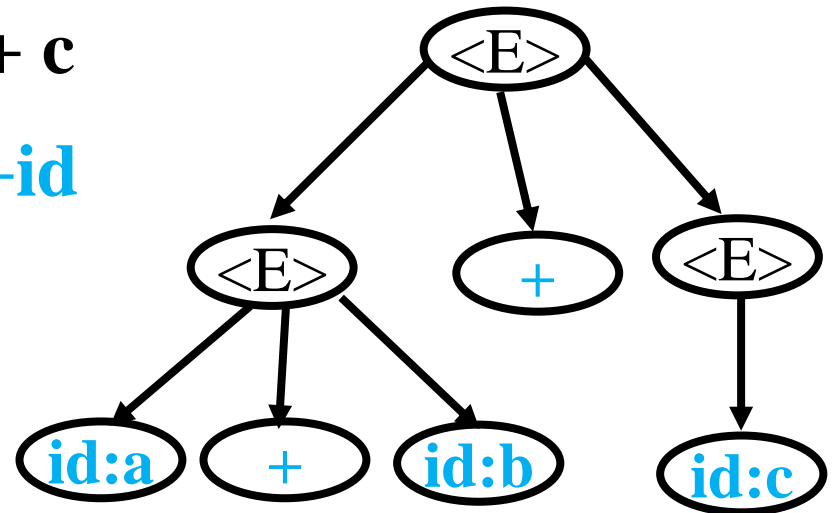
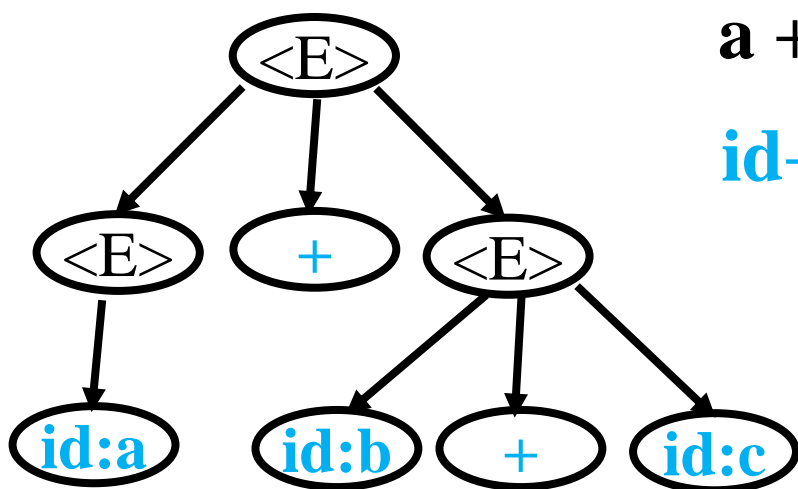
id := num ; id := num + num



Ambiguous Grammars

- Two leftmost derivations
- Two rightmost derivations
- Two parse trees

- 1 $\langle E \rangle \rightarrow \langle E \rangle + \langle E \rangle$
- 2 $\langle E \rangle \rightarrow \langle E \rangle * \langle E \rangle$
- 3 $\langle E \rangle \rightarrow id$
- 4 $\langle E \rangle \rightarrow (\langle E \rangle)$



Non Ambiguous Grammars for Arithmetic Expressions

Ambiguous grammar:

- 1 $\langle E \rangle \rightarrow \langle E \rangle + \langle E \rangle$
- 2 $\langle E \rangle \rightarrow \langle E \rangle * \langle E \rangle$
- 3 $\langle E \rangle \rightarrow \mathbf{id}$
- 4 $\langle E \rangle \rightarrow (\langle E \rangle)$

Non-Ambiguous grammar:

- 1 $\langle E \rangle \rightarrow \langle E \rangle + \langle T \rangle$
- 2 $\langle E \rangle \rightarrow \langle T \rangle$
- 3 $T \rightarrow \langle T \rangle * \langle F \rangle$
- 4 $T \rightarrow \langle F \rangle$
- 5 $F \rightarrow \mathbf{id}$
- 6 $F \rightarrow (\langle E \rangle)$

Non Ambiguous Grammars for Arithmetic Expressions

Non-Ambiguous grammar:

1 $\langle E \rangle \rightarrow \langle E \rangle + \langle T \rangle$

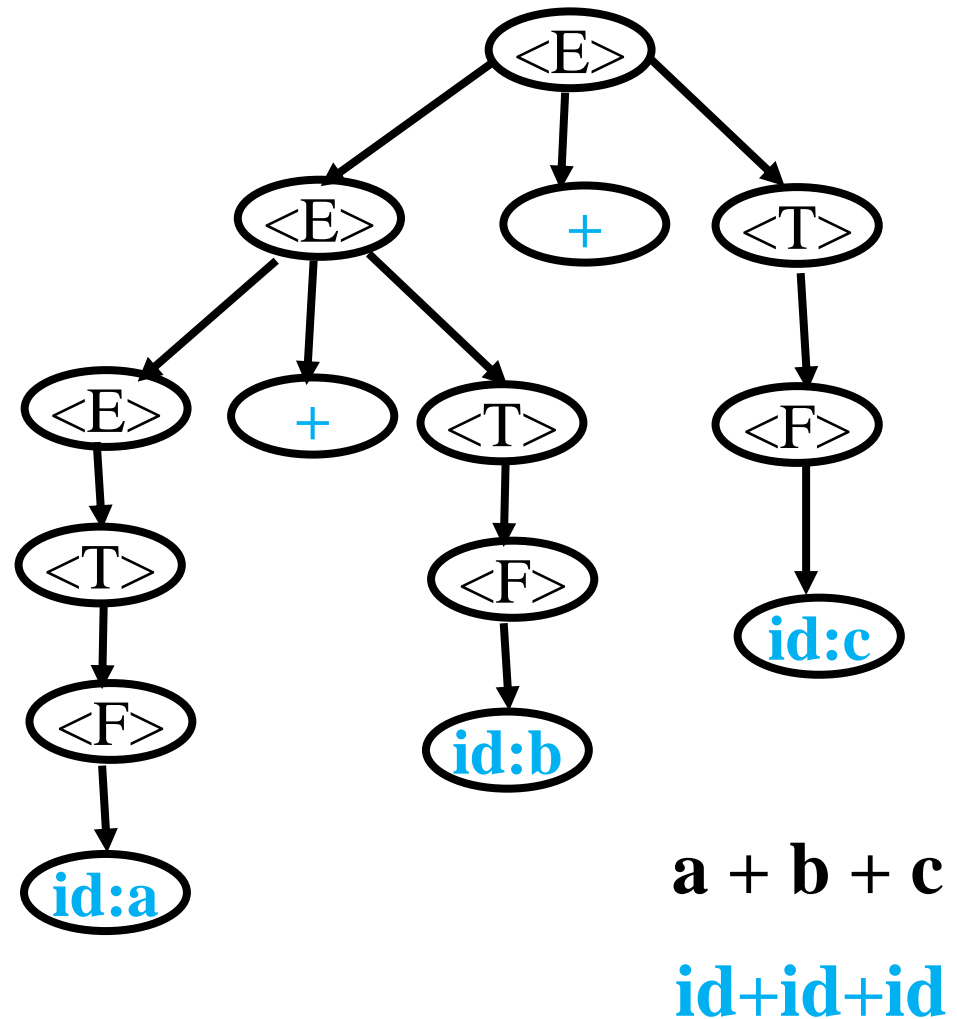
2 $\langle E \rangle \rightarrow \langle T \rangle$

3 $T \rightarrow \langle T \rangle * \langle F \rangle$

4 $T \rightarrow \langle F \rangle$

5 $F \rightarrow \text{id}$

6 $F \rightarrow (\langle E \rangle)$



Predictive Parsing – LL(1)

- Left to right, Left most derivation, 1 token look ahead
- We must uniquely predict the next rule by the current non-terminal and 1 next token
 - Grammar must be non ambiguous
 - Grammar must not contain left recursion (right recursion is ok)
 - Grammar must be left factored
 - These conditions are necessary but not sufficient

Example Predictive Parser

```
<S> → id := <E>
<S> → if (<E>) <S> else <S>
<E> → <T> <EP>
<T> → id | (<E>)
<EP> → ε | + <E>
```

```
def parse_S():
    if next(id):
        match(id)
        match(assign)
        parse_E()
    elif next(if_tok):
        match(if_tok)
        match(lp)
        parse_E()
        match(rp)
        parse_S()
        match(else_tok)
        parse_S()
    else:
        syntax_error()
```

```
def parse_E():
    parse_T()
    parse_EP()

def parse_T():
    if next(id):
        match(id)
    elif next(lp):
        match(lp)
        parse_E()
        match(rp)
    else:
        syntax_error()
```

```
def parse_EP():
    if next(plus):
        match(plus)
        parse_E()
    elif (next(rp) or
          next(else_tok) or
          next(eof)):
        return # ε
    else:
        syntax_error()
```

Example Predictive Parser

```
<S> → id := <E>
<S> → if (<E>) <S> else <S>
<E> → <T> <EP>
<T> → id | (<E>)
<EP> → ε | + <E>
```

```
<S> → <S> ; <S> ?
```

```
def parse_S():
    if next(id):
        match(id)
        match(assign)
        parse_E()
    elif next(if_tok):
        match(if_tok)
        match(lp)
        parse_E()
        match(rp)
        parse_S()
        match(else_tok)
        parse_S()
    else:
        syntax_error()
```

```
def parse_E():
    parse_T()
    parse_EP()

def parse_T():
    if next(id):
        match(id)
    elif next(lp):
        match(lp)
        parse_E()
        match(rp)
    else:
        syntax_error()
```

```
def parse_EP():
    if next(plus):
        match(plus)
        parse_E()
    elif (next(rp) or
          next(else_tok) or
          next(eof)):
        return # ε
    else:
        syntax_error()
```

Nullable Non-Terminals

- Non-terminal A is **nullable** iff $A \rightarrow^* \varepsilon$

Example:

```
<S> → id := <E>
<S> → if (<E>) <S> else <S>
<E> → <T> <EP>
<T> → id | (<E>)
<EP> → ε | + <E>
```

- Is S nullable?
- Is E nullable?
- Is T nullable?
- Is EP nullable?

Computing Nullable

- Non-terminal A **nullable** iff $A \rightarrow^* \varepsilon$

Nullable = \emptyset

for each rule $A \rightarrow \varepsilon$:

 add A to Nullable

while changes occur do:

 for each rule $A \rightarrow A_1 A_2 \dots A_n$:

 if $\{A_1, A_2, \dots, A_n\} \subseteq \text{Nullable}$:

 add A to Nullable

Example – Computing Nullable

```
<S> → id := <E>  
<S> → if (<E>) <S> else <S>  
<E> → <T> <EP>  
<T> → id | (<E>)  
<EP> → ε | + <E>
```

```
Nullable = ∅  
for each rule A → ε:  
    add A to Nullable  
while changes occur do:  
    for each rule A → A1 A2 ... An :  
        if {A1, A2, ..., An} ⊆ Nullable:  
            add A to Nullable
```

Nullable = ∅

Nullable = {EP}

Fix point reached

First Sets

- $\text{First}(\alpha)$ – set of all tokens that can be the start of α
- $\text{First}(\alpha) = \{ t \mid \exists \beta: \alpha \rightarrow^* t \beta \}$

Example:

```
<S> → id := <E>  
<S> → if (<E>) <S> else <S>  
<E> → <T> <EP>  
<T> → id | (<E>)  
<EP> → ε | + <E>
```

- $\text{First}(S) =$
- $\text{First}(E) =$
- $\text{First}(T) =$
- $\text{First}(EP) =$

Computing First

- $\text{First}(\alpha)$ – set of all tokens that can be the start of α
- $\text{First}(\alpha) = \{ t \mid \exists \beta: \alpha \rightarrow^* t \beta \}$

For each token t : $\text{First}(t) = \{t\}$

For each non-terminal A : $\text{First}(A) = \emptyset$

while changes occur do:

 for each rule $A \rightarrow A_1 A_2 \dots A_n$:

 for each $1 \leq i \leq n$:

 if $\{A_1, A_2, \dots, A_{i-1}\} \subseteq \text{Nullable}$:

 add $\text{First}(A_i)$ to $\text{First}(A)$

- For convenience, we defined $\text{First}(t) = \{t\}$ for tokens

Example – Computing First

```
1 <S> → id := <E>
2 <S> → if (<E>) <S> else <S>
3 <E> → <T> <EP>
4,5 <T> → id | (<E>)
6,7 <EP> → ε | + <E>
```

Nullable = {EP}

For each token t : $\text{First}(t) = \{t\}$
For each non-terminal A : $\text{First}(A) = \emptyset$
while changes occur do:
 for each rule $A \rightarrow A_1 A_2 \dots A_n$:
 for each $1 \leq i \leq n$:
 if $\{A_1, A_2, \dots, A_{i-1}\} \subseteq \text{Nullable}$:
 add $\text{First}(A_i)$ to $\text{First}(A)$

$\text{First}(t) = \{t\}$ for t in {id, ass, if, lp, rp, else, plus}

```
add id to First(S)      # 1
add if to First(S)     # 2
                        # 3
add id to First(T)     # 4
add lp to First(T)     # 5
                        # 6
add plus to First(EP)  # 7

add id,lp to first(E)  # 3
```

A	First(A)
S	id, if
E	id, lp
T	id, lp
EP	plus

Fix point reached

Follow Sets

- Follow(A) – set of all tokens that can appear after A
- Follow(A) = { $t \mid \exists \beta, \gamma: \langle S \rangle \rightarrow^* \beta \langle A \rangle t \gamma$ }

Example:

```
<S> → id := <E>  
<S> → if (<E>) <S> else <S>  
<E> → <T> <EP>  
<T> → id | (<E>)  
<EP> → ε | + <E>
```

- Follow(S) =
- Follow(E) =
- Follow(T) =
- Follow(EP) =

Computing Follow

- Follow(A) – set of all tokens that can appear after A
- Follow(A) = { t | $\exists \beta, \gamma: \langle S \rangle \rightarrow^* \beta \langle A \rangle t \gamma$ }

For each non-terminal A: Follow(A) = \emptyset

For the start non-terminal S: Follow(S) = {eof}

while changes occur do:

 for each rule $\langle A \rangle \rightarrow A_1 A_2 \dots A_n$:

 for each $1 \leq i \leq n$:

 if $\{A_{i+1}, \dots, A_n\} \subseteq \text{Nullable}$:

 add Follow(A) to Follow(A_i)

 for each $1 \leq i < j \leq n$:

 if $\{A_{i+1}, \dots, A_{j-1}\} \subseteq \text{Nullable}$:

 add First(A_j) to Follow(A_i)

Example – Computing Follow

```
1 <S> → id := <E>
2 <S> → if (<E>) <S> else <S>
3 <E> → <T> <EP>
4,5 <T> → id | (<E>)
6,7 <EP> → ε | + <E>
```

```
Nullable = {EP}
First(S) = {id, if}
First(E) = First(T) = {id, lp}
First(EP) = {plus}
```

```
Follow(A) = ∅ for A in {E, T, EP}
Follow(S) = {eof}
```

```
add eof to Follow(E)      # 1
add rp to Follow(E)      # 2
add else to Follow(S)    # 2
add eof, rp to Follow(T) # 3
add eof, rp to Follow(EP) # 3
add plus to Follow(T)    # 3

add else to Follow(E)    # 1
add else to Follow(T)    # 3
add else to Follow(EP)   # 3
```

Fix point reached

```
For each non-terminal A≠S: Follow(A) = ∅
For the start non-terminal: Follow(S) = {eof}
while changes occur do:
  for each rule <A> → A1 A2 ... An :
    for each 1 ≤ i ≤ n:
      if {Ai+1, ..., An} ⊆ Nullable:
        add Follow(A) to Follow(Ai)
    for each 1 ≤ i < j ≤ n:
      if {Ai+1, ..., Aj-1} ⊆ Nullable:
        add First(Aj) to Follow(Ai)
```

A	Follow(A)
S	eof, else
E	eof, rp, else
T	eof, rp, plus, else
EP	eof, rp, else

Select Sets

- For each rule $A \rightarrow \alpha$:
 - If α is not nullable: $\text{Select}(A \rightarrow \alpha) = \text{First}(\alpha)$
 - If α is nullable: $\text{Select}(A \rightarrow \alpha) = \text{First}(\alpha) \cup \text{Follow}(A)$
- If we need to parse A , we can decide which rule to apply according to the next token and the select sets
- The grammar is LL(1) iff:
for every two grammar rules $\langle A \rangle \rightarrow \alpha$ and $\langle A \rangle \rightarrow \beta$:
 $\text{Select}(A \rightarrow \alpha) \cap \text{Select}(A \rightarrow \beta) = \emptyset$

Example – Computing Select

$\langle S \rangle \rightarrow \mathbf{id} := \langle E \rangle$
 $\langle S \rangle \rightarrow \mathbf{if} (\langle E \rangle) \langle S \rangle \mathbf{else} \langle S \rangle$
 $\langle E \rangle \rightarrow \langle T \rangle \langle EP \rangle$
 $\langle T \rangle \rightarrow \mathbf{id} \mid (\langle E \rangle)$
 $\langle EP \rangle \rightarrow \varepsilon \mid + \langle E \rangle$

A	Nullable	First(A)	Follow(A)
S	no	id, if	eof, else
E	no	id, lp	eof, rp, else
T	no	id, lp	eof, rp, plus, else
EP	yes	plus	eof, rp, else

For each rule $A \rightarrow \alpha$:

If α is not nullable:

$$\text{Select}(A \rightarrow \alpha) = \text{First}(\alpha)$$

If α is nullable:

$$\text{Select}(A \rightarrow \alpha) = \text{First}(\alpha) \cup \text{Follow}(A)$$

$A \rightarrow \alpha$	Select($A \rightarrow \alpha$)
$\langle S \rangle \rightarrow \mathbf{id} := \langle E \rangle$	id
$\langle S \rangle \rightarrow \mathbf{if} (\langle E \rangle) \langle S \rangle \mathbf{else} \langle S \rangle$	if
$\langle E \rangle \rightarrow \langle T \rangle \langle EP \rangle$	id, lp
$\langle T \rangle \rightarrow \mathbf{id}$	id
$\langle T \rangle \rightarrow (\langle E \rangle)$	lp
$\langle EP \rangle \rightarrow \varepsilon$	eof, rp, else
$\langle EP \rangle \rightarrow + \langle E \rangle$	plus

Generating a Predictive Parser

- Compute: Nullable, First, Follow, Select
 - If the grammar is not LL(1) report an error
- Create a procedure for every non-terminal A:
 - For every rule $A \rightarrow \alpha$:
 - If the next token is in $\text{Select}(A \rightarrow \alpha)$,
apply the rule $\langle A \rangle \rightarrow \alpha$: this contains recursive
calls to procedures of other non-terminals
 - If the next token is not in any $\text{Select}(A \rightarrow \alpha)$, report
syntax error

Putting It All Together

$A \rightarrow \alpha$	Select($A \rightarrow \alpha$)
$\langle S \rangle \rightarrow \text{id} := \langle E \rangle$	id
$\langle S \rangle \rightarrow \text{if} (\langle E \rangle) \langle S \rangle \text{ else } \langle S \rangle$	if
$\langle E \rangle \rightarrow \langle T \rangle \langle EP \rangle$	id, lp
$\langle T \rangle \rightarrow \text{id}$	id
$\langle T \rangle \rightarrow (\langle E \rangle)$	lp
$\langle EP \rangle \rightarrow \varepsilon$	eof, rp, else
$\langle EP \rangle \rightarrow + \langle E \rangle$	plus

```
def parse_S():
    if next(id):
        match(id)
        match(assign)
        parse_E()
    elif next(if_tok):
        match(if_tok)
        match(lp)
        parse_E()
        match(rp)
        parse_S()
        match(else_tok)
        parse_S()
    else:
        syntax_error()
```

```
def parse_E():
    parse_T()
    parse_EP()

def parse_T():
    if next(id):
        match(id)
    elif next(lp):
        match(lp)
        parse_E()
        match(rp)
    else:
        syntax_error()
```

```
def parse_EP():
    if next(plus):
        match(plus)
        parse_E()
    elif (next(rp) or
          next(else_tok) or
          next(eof)):
        return #  $\varepsilon$ 
    else:
        syntax_error()
```