CS 314 Principles of Programming Languages

Lecture 7: LL(1) Parsing

Prof. Zheng Zhang

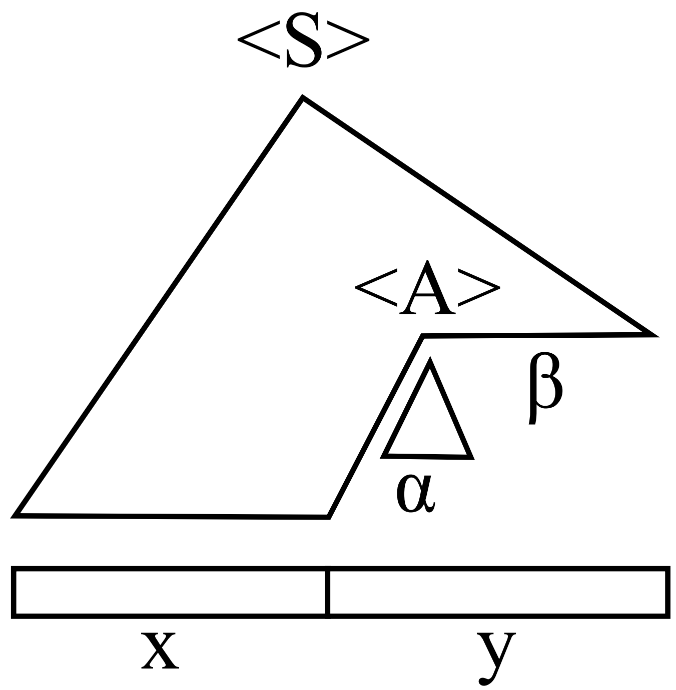
Rutgers University

September 26, 2018

Class Information

- Homework 1 and 2 are being graded.
- Homework 3 will be posted by the end of today.

Review: Top-Down Parsing - LL(1)



Basic Idea:

- The parse tree is constructed from the root, expanding non-terminal nodes on the tree's frontier following a **leftmost** derivation.
- The input program is read from **left** to right, and input tokens are read (consumed) as the program is parsed.
- The next non-terminal symbol is replaced using one of its rules. The particular choice has to be unique and uses parts of the input (partially parsed program), for instance the first token of the remaining input.

Review: Predictive Parsing

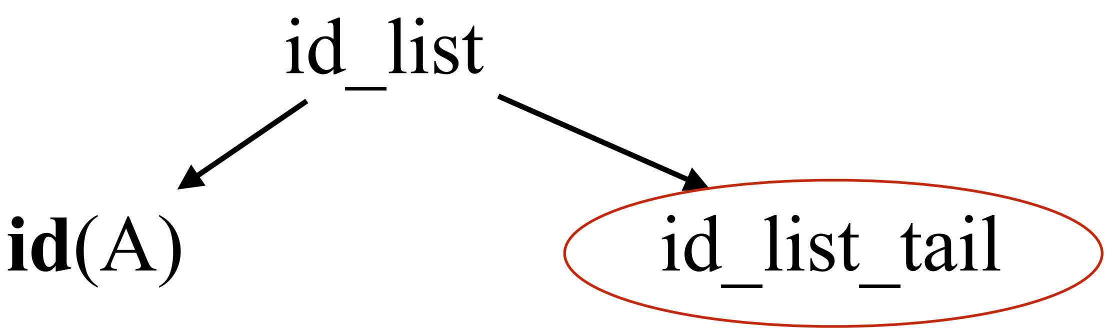
Basic idea:

For any two productions A ::= α and A ::= β, we would like *a distinct way of choosing the correct production to expand.*

Revisiting the id_list Example

id_list ::= **id** id_list_tail
id_list_tail ::= **, id** id_list_tail
id_list_tail ::= **;**

Remaining Input:
, B , C ;

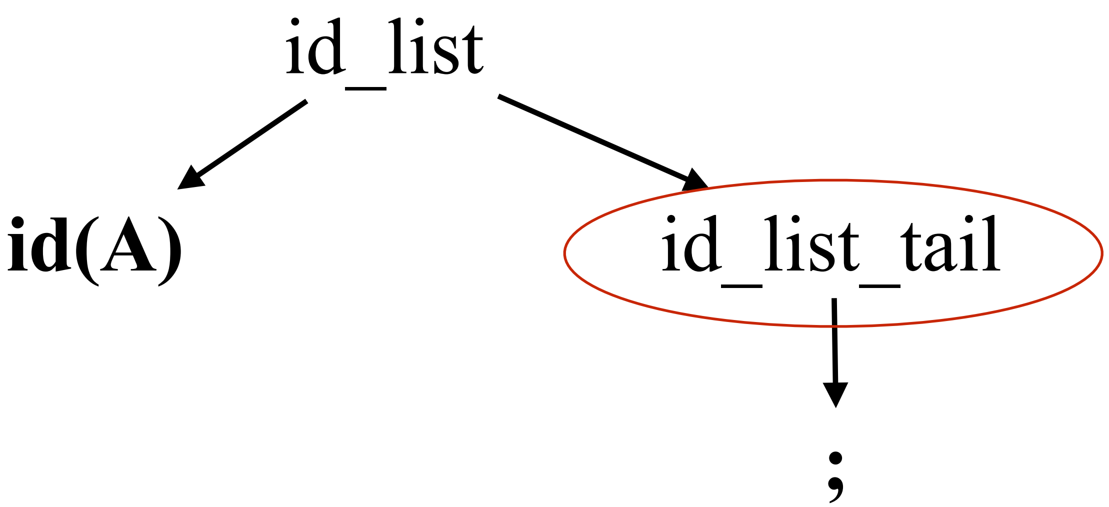


Applied Production:

Revisiting the id_list Example

id_list ::= **id** id_list_tail
id_list_tail ::= **, id** id_list_tail
id_list_tail ::= **;**

Remaining Input:
$\boxed{,}$ B , C ;

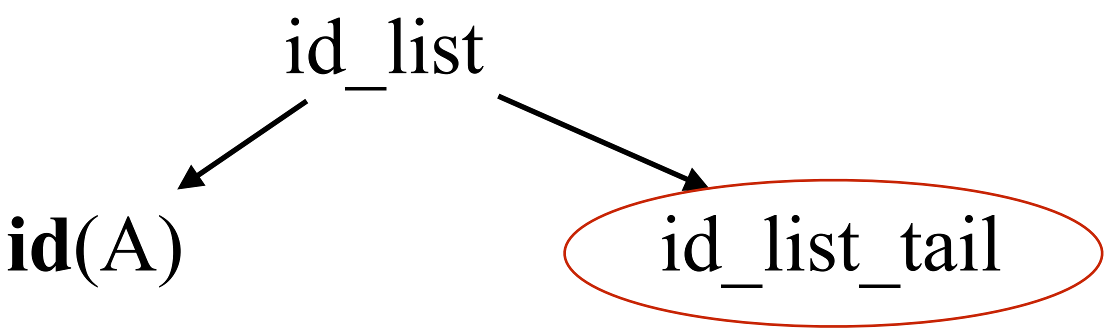


Mismatch!

Applied Production:
~~id_list_tail ::= ;~~

Revisiting the id_list Example

id_list ::= **id** id_list_tail
id_list_tail ::= **, id** id_list_tail
id_list_tail ::= **;**



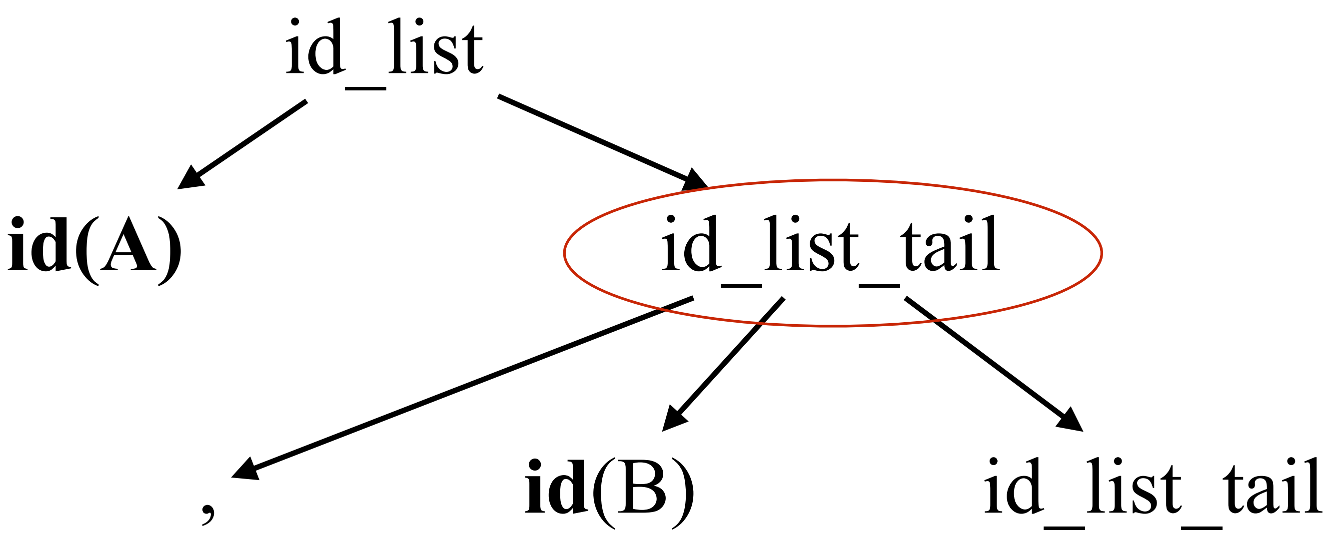
Remaining Input:
, B , C ;

Applied Production:

Revisiting the id_list Example

id_list ::= **id** id_list_tail
id_list_tail ::= **, id** id_list_tail
id_list_tail ::= **;**

Remaining Input:
, B , C ;

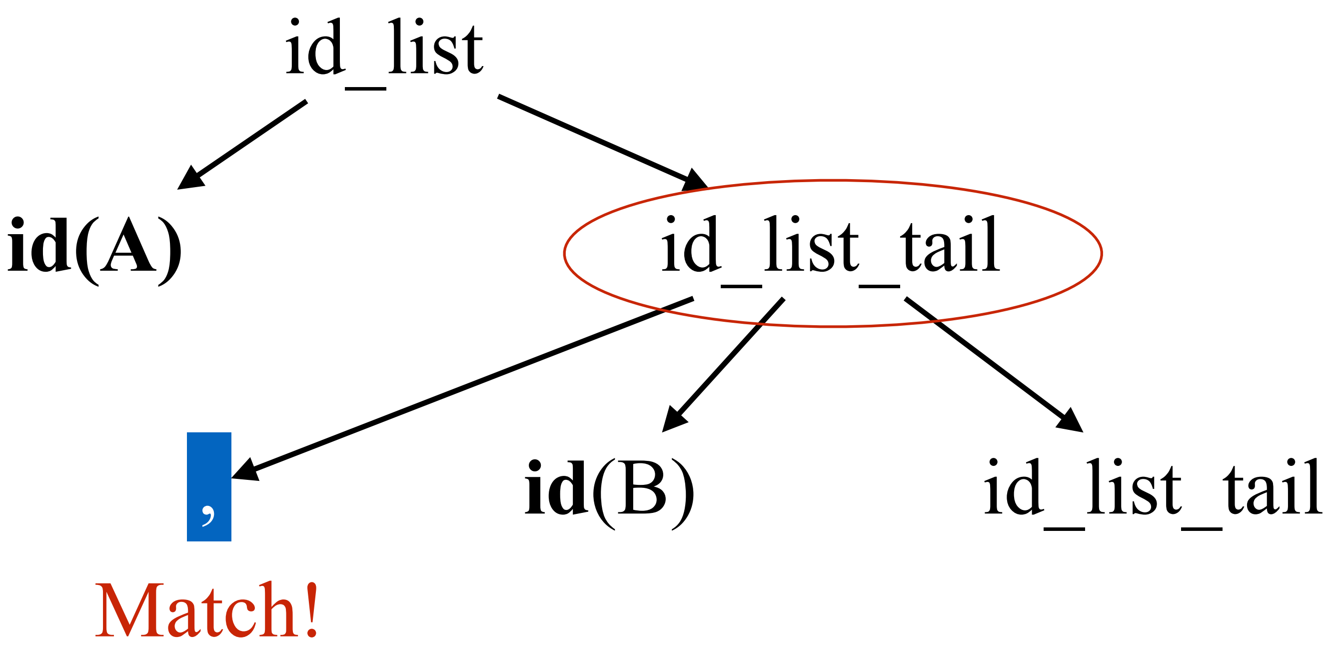


Applied Production:
id_list_tail ::= **, id** id_list_tail

Revisiting the id_list Example

id_list ::= **id** id_list_tail
id_list_tail ::= **, id** id_list_tail
id_list_tail ::= **;**

Remaining Input:
$\boxed{,}$ B , C ;



Applied Production:
id_list_tail ::= **, id** id_list_tail

Review: First Set

For some string α, define **FIRST**(α) as the set of tokens that appear as the first symbol in some string derived from α.

That is

$x \in \text{FIRST}(\alpha)$ iff $\alpha \Rightarrow^* \mathbf{x}\gamma$ for some string γ

Review: Predictive Parsing

Key Property:

Whenever two productions A ::= α and A ::= β both appear in the grammar, we would like

- $FIRST(\alpha) \cap FIRST(\beta) = \varnothing$

Revisiting the id_list Example

id_list ::= **id** id_list_tail
id_list_tail ::= **, id** id_list_tail
id_list_tail ::= **;**

Remaining Input:
, B , C ;

id_list

id(A) id_list_tail

FIRST(**, id** id_list_tail) = { , }

FIRST(**;**) = { ; }

FIRST (**, id** id_list_tail ∩ *FIRST* (;) = ∅

Given id_list_tail as the first **non-terminal** to expand in the tree:

If the first token of remaining input is , we choose the rue

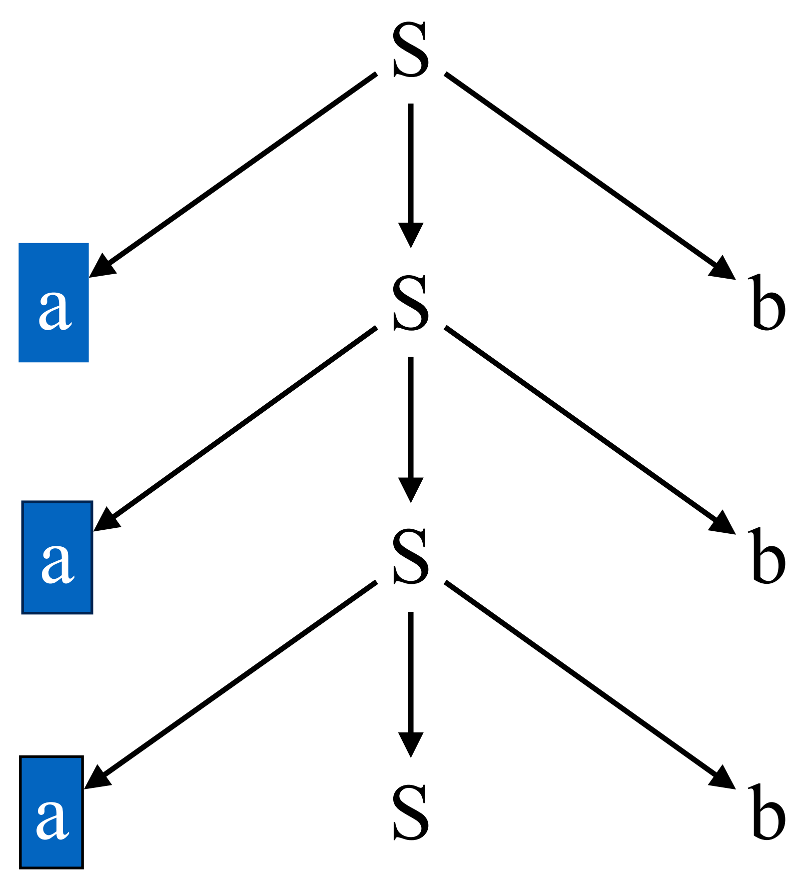
id_list_tail ::= **, id** id_list_tail

If the first token of remaining input is ; we choose the rule

id_list_tail ::= **;**

Revisiting the LL(1) Parsing Example

S ::= a S b | ε

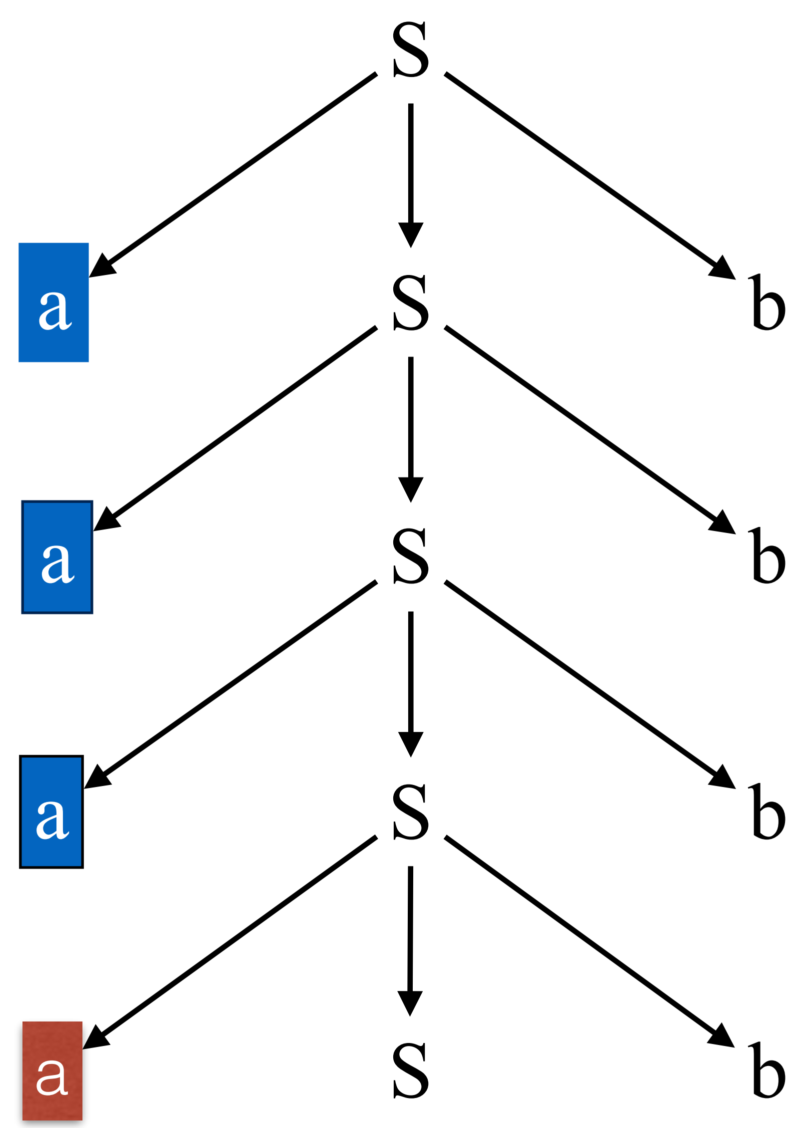


Remaining Input:
b b b

Applied Production:

Revisiting the LL(1) Parsing Example

S ::= a S b | ε



Remaining Input:
b b b

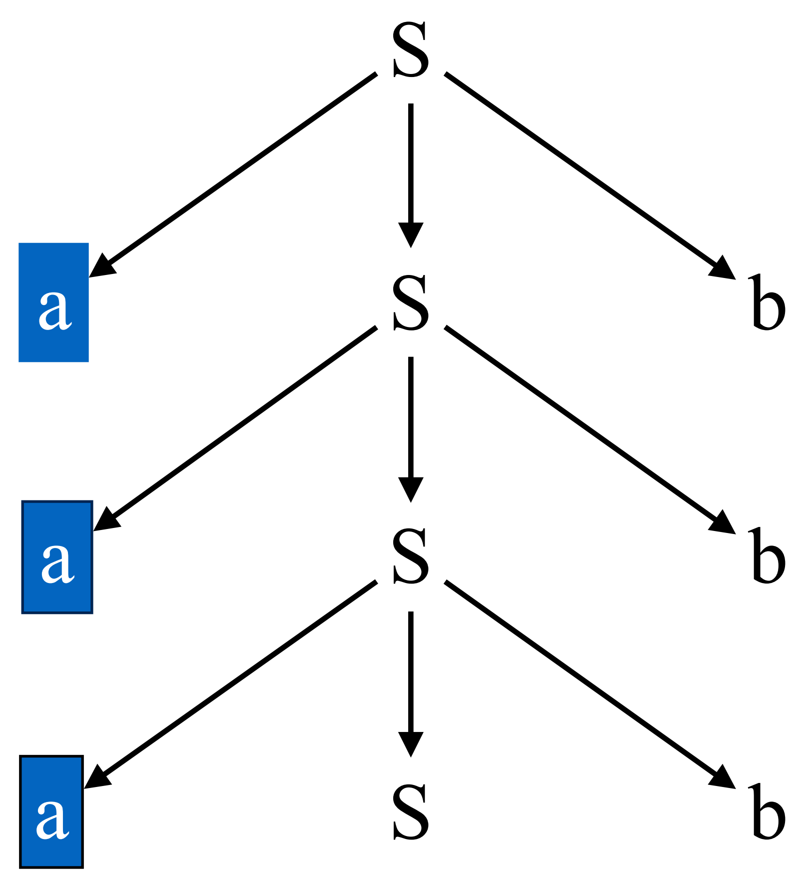
Applied Production:

~~S ::= a S b~~

Mismatch!
It only means S ::= aSb is not the right production rule to use!

Revisiting the LL(1) Parsing Example

S ::= a S b | ε

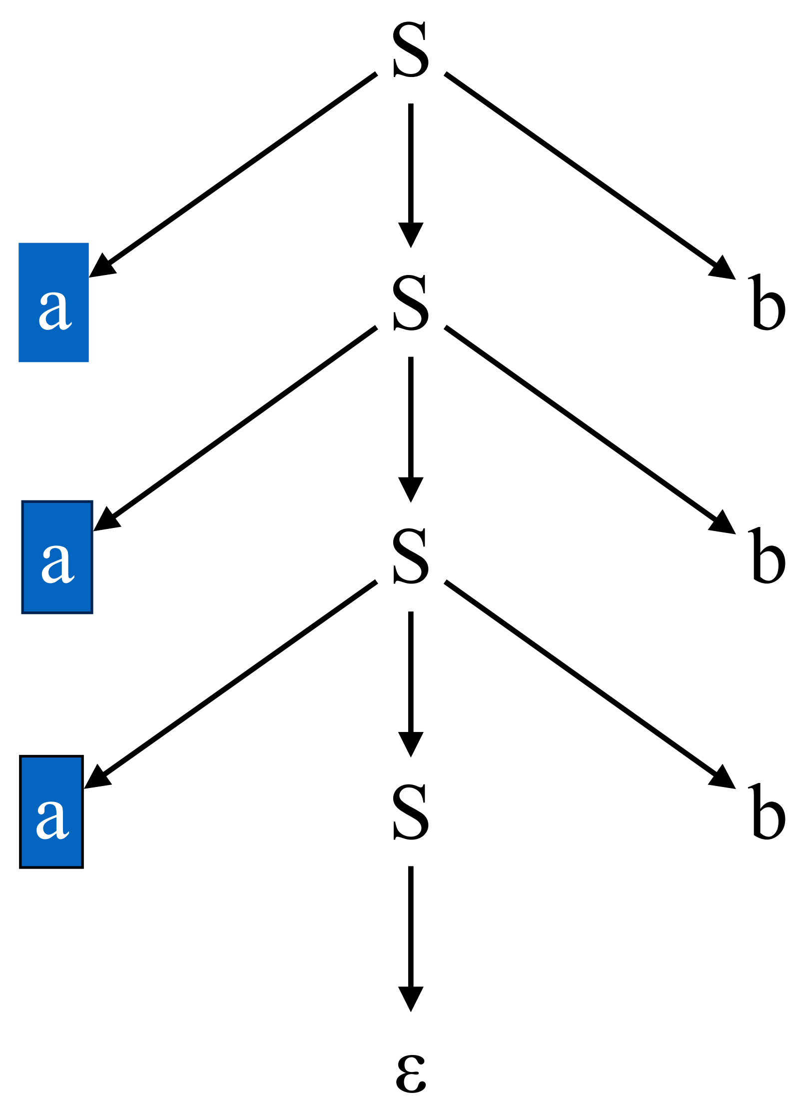


Remaining Input:
b b b

Applied Production:

Revisiting the LL(1) Parsing Example

S ::= a S b | ε



Remaining Input:
b b b

Applied Production:
S ::= ε

S ::= ε turns out to be the right rule later.

However, at this point, ε does not match "b" either !

Review: Follow Set

For a non-terminal A, define **FOLLOW**(A) as the set of terminals that can appear immediately to the right of A in some sentential form.

Thus, a non-terminal's **FOLLOW** set specifies the tokens that can legally appear after it. A terminal symbol has no **FOLLOW** set.

FIRST and FOLLOW sets can be constructed automatically

Review: Predictive Parsing

Key Property:

Whenever two productions $A ::= \alpha$ and $A ::= \beta$ both appear in the grammar, we would like

- $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$, and
- if $\alpha \Rightarrow^* \varepsilon$, then $FIRST(\beta) \cap FOLLOW(A) = \emptyset$

Analogue case for $\beta \Rightarrow^* \varepsilon$.

Note: due to first condition, at most one of α and β can derive ε.

This would allow the parser to make a correct choice with a lookahead of only one symbol!

Review: LL(1) Grammar

Define *PREDICT*(A ::= δ) for rule A ::= δ

- *FIRST* (δ) - { ε } U Follow (A), if ε ∈ *FIRST*(δ)
- *FIRST* (δ) otherwise

A Grammar is LL(1) iff
(A ::= α and A ::= β) implies

PREDICT(A ::= α) ∩ PREDICT(A ::= β) = ∅

Table Driven LL(1) Parsing

Example:

S ::= **a** S **b** | ε

Predict Sets

PREDICT(S ::= aSb) = {a}

PREDICT(S ::= ε) ={b, eof}

LL(1) parse table

	a	b	eof	other
S	S ::= aSb	S ::= ε	S ::= ε	error

Table Driven LL(1) Parsing

Example:

S ::= **a** S **b** | ε

Predict Sets

$PREDICT$(S ::= aSb) = {a}
$PREDICT$(S ::= ε) ={**b**, eof}

LL(1) parse table

	a	b	eof	other
S	S ::= aSb	S ::= ε	S ::= ε	error

Review: Table Driven LL(1) Parsing

Input: *a string w and a parsing table M for G*

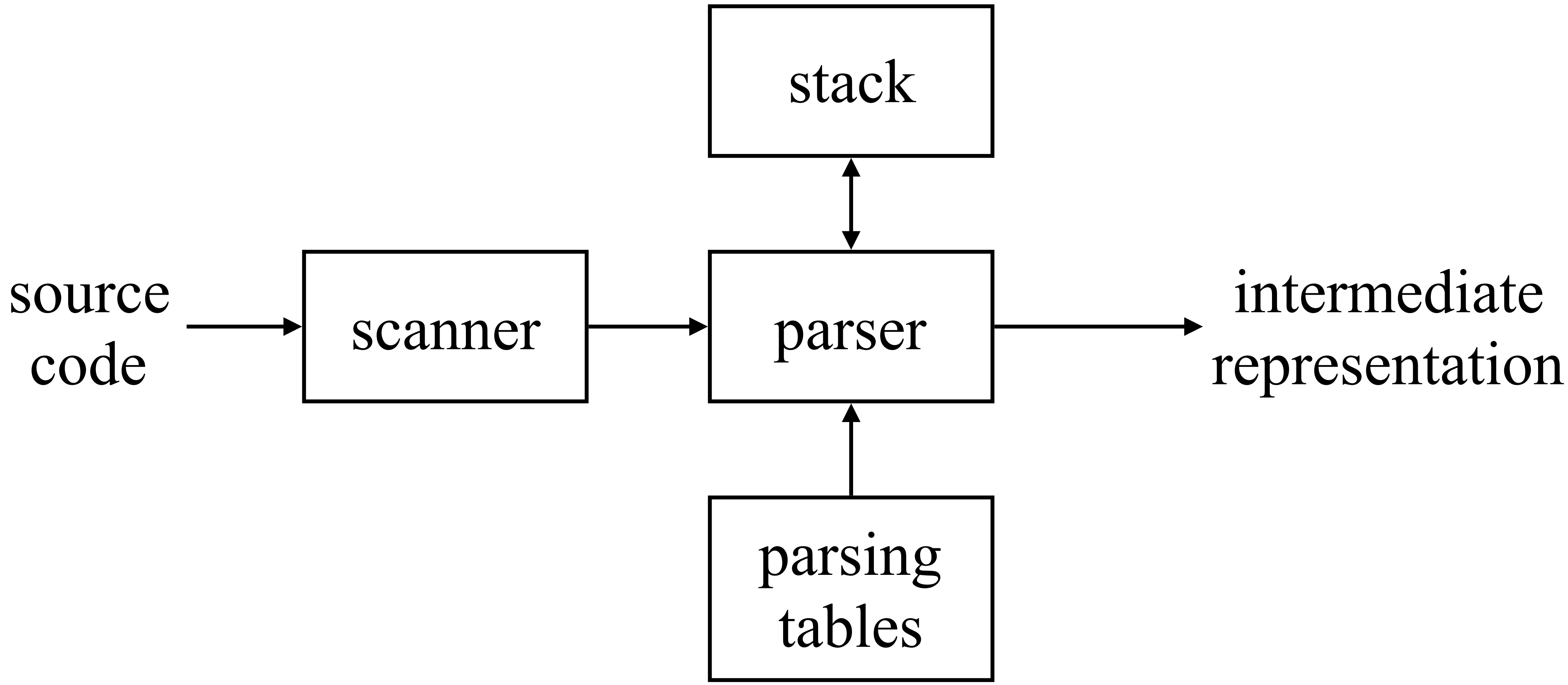
push eof
push Start Symbol
token ← *next_token*()
X ← top-of-stack
repeat
    if X is a terminal then
        if X == token then
            pop X
            token ← *next_token*()
        else error()
    else /* X is a non-terminal */
        if **M**[X, token] == $X \rightarrow Y_1 Y_2 \ldots Y_k$ then
            pop X
            push $Y_k, Y_{k-1}, \ldots, Y_1$
        else error()
    X ← top-of-stack
until X = EOF
if token != EOF then error()

	a	b	eof	other
S	S ::= aSb	S ::= ε	S ::= ε	error

M is the parse table

Predictive Parsing

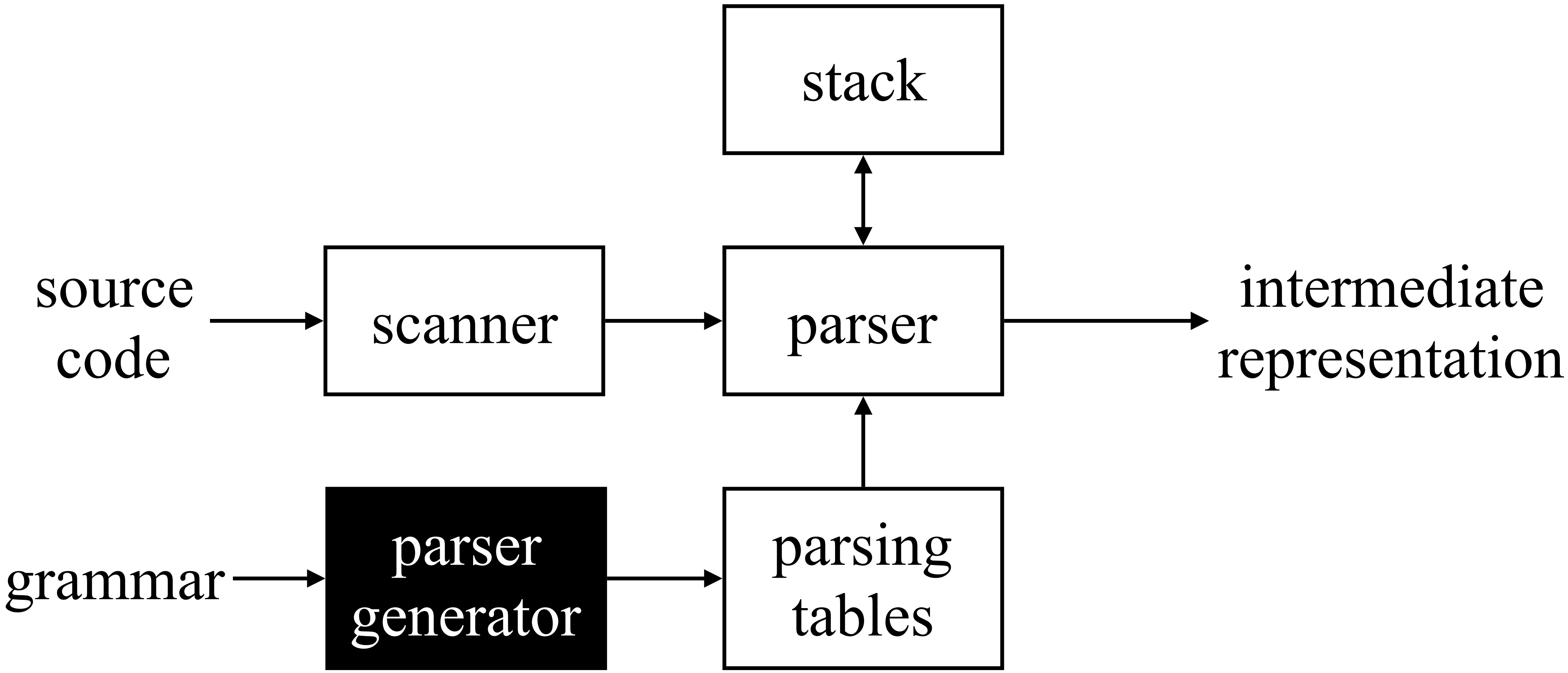
Now, a predictive parser looks like:



Rather than writing code, we build tables.

Predictive Parsing

Now, a predictive parser looks like:



Rather than writing code, we build tables.
Building tables can be automated!

Predictive Parsing

So far:

- Introduced **FIRST**, **FOLLOW**, and **PREDICT** sets
- Introduced **LL(1)** condition:
  - A grammar G can be parsed predictively with one symbol of lookahead if for all pairs of productions A ::= α and A ::= β that satisfy:
    PREDICT(A ::= α) $\cap$ PREDICT(A ::= β) = $\varnothing$
- Introduced a recursive descent parser for an **LL(1)** grammar

How to automatically construct ***FIRST*** and ***FOLLOW*** sets?

FIRST and FOLLOW Sets

FIRST(α):

For some $\alpha \in (T \cup NT \cup EOF \cup \varepsilon)^*$, define **FIRST** ($\alpha$) as the set of tokens that appear as the first symbol in some string that derives from α.

That is, $\mathbf{x} \in FIRST(\alpha)$ iff $\alpha \Rightarrow^* \mathbf{x}\gamma$ for some γ

FIRST set is defined over the strings of grammar symbols $(T \cup NT \cup EOF \cup \varepsilon)^*$

T: terminals   NT: non-terminals

Computing *FIRST* Sets

For a production $A \rightarrow B_1 B_2 \ldots B_k$:

- FIRST(A) includes FIRST(B_1) - ε
- FIRST(A) includes FIRST(B_2) - ε if B_1 can be rewritten as ε
- FIRST(A) includes FIRST(B_3) - ε if both B_1 *and* B_2 can derive ε
- …
- FIRST(A) includes FIRST(B_m) - ε if $B_1 B_2 \ldots B_{m-1}$ can derive ε

FIRST(A) includes FIRST(B_1) … FIRST(B_m) not including ε iff
$\varepsilon \in$ FIRST(B_1), FIRST(B_2), FIRST(B_3), …, FIRST(B_{m-1})

FIRST(A) includes ε iff
$\varepsilon \in$ FIRST(B_1), FIRST(B_2), FIRST(B_3), …, FIRST(B_k)

First Set Construction

Build FIRST(X) for all grammar symbols X:

- For each X as a terminal, then FIRST(X) is $\{X\}$
- If $X ::= \varepsilon$, then $\varepsilon \in$ FIRST(X)
- 1. For each X as a non-terminal, initialize FIRST(X) to $\varnothing$

  2. ***Iterate until*** no more terminals or ε can be added to any FIRST(X):

     For each rule in the grammar of the form $X ::= Y_1 Y_2 \ldots Y_k$

       add a to FIRST(X) if $a \in$ FIRST(Y_1)

       add a to FIRST(X) if $a \in$ FIRST(Y_i) and $\varepsilon \in$ FIRST(Y_j) for all $1 \leq j \leq i-1$ and $i \geq 2$

       add ε to FIRST(X) if $\varepsilon \in$ FIRST(Y_i) for all $1 \leq i \leq k$

     End iterate

Filling in the Details: Computing *FIRST* sets

for each $x \in$ (T ∪ EOF ∪ ε)
    FIRST(x) ← {x}
for each A ∈ **NT**, ***FIRST***(A) ← ∅

while (***FIRST*** sets are still changing) do
    for each p ∈ P, of the form $X \rightarrow Y_1Y_2\ldots Y_k$ do
    temp ← ***FIRST***(Y_1) - { ε }
        i ← 1
        while (i ≤ k-1 and ε ∈ ***FIRST***(Y_i))
            temp ← temp ∪ (***FIRST***(Y_{i+1}) - { ε })
            i ← i + 1
        end // while loop
        if i == k and ε ∈ ***FIRST***(Y_k)
        then temp ← temp ∪ { ε }
        FIRST(X) ← ***FIRST***(X) ∪ temp
      end // if - then
    end // for loop
end // while loop

Initially, set ***FIRST*** for each terminal symbol, EOF and ε

Filling in the Details: Computing *FIRST* sets

ε complicates matters

```
for each x ∈ ( T ∪ EOF ∪ ε)
    FIRST(x) ← {x}
for each A ∈ NT, FIRST(A) ← ∅

while (FIRST sets are still changing) do
    for each p ∈ P, of the form X → Y1Y2…Yk do
    temp ← FIRST(Y1) - { ε }
        i ← 1
        while ( i ≤  k-1 and ε ∈ FIRST(Yi) )
           temp ← temp ∪ (FIRST(Yi+1) - { ε })
           i ← i + 1
        end // while loop
        if i == k and ε ∈ FIRST(Yk)
             then temp ← temp ∪ { ε }
        FIRST(X) ← FIRST(X) ∪ temp
      end // if - then
    end // for loop
end // while loop
```

If ***FIRST***(Y_1) contains ε, then we need to add ***FIRST***(Y_2) to rhs, and …

Filling in the Details: Computing *FIRST* sets

ε complicates matters

for each $x \in$ (T ∪ EOF ∪ ε)
    FIRST(x) ← {x}
for each A ∈ **NT**, ***FIRST***(A) ← ∅

while (***FIRST*** sets are still changing) do
    for each p ∈ P, of the form X → $Y_1 Y_2 \ldots Y_k$ do
    temp ← ***FIRST***(Y_1) - { ε }
        i ← 1
        while (i ≤ k-1 and ε ∈ ***FIRST***(Y_i))
            temp ← temp ∪ (***FIRST***(Y_{i+1}) - { ε })
            i ← i + 1
        end // while loop
        if i == k and ε ∈ ***FIRST***(Y_k)
            then temp ← temp ∪ { ε }
        FIRST(X) ← ***FIRST***(X) ∪ temp
      end // if - then
    end // for loop
end // while loop

If the entire rhs can go to ε, then we add ε to ***FIRST***(lhs)

Computing *FIRST* sets

```
for each x ∈ ( T ∪ EOF ∪ ε)
    FIRST(x) ← {x}
for each A ∈ NT, FIRST(A) ← ∅

while (FIRST sets are still changing) do
    for each p ∈ P, of the form X → Y1Y2…Yk do
    temp ← FIRST(Y1) - { ε }
        i ← 1
        while ( i ≤ k-1 and ε ∈ FIRST(Yi) )
          temp ← temp ∪ (FIRST(Yi+1) - { ε })
          i ← i + 1
        end // while loop
        if i == k and ε ∈ FIRST(Yk)
            then temp ← temp ∪ { ε }
        FIRST(X) ← FIRST(X) ∪ temp
      end // if - then
    end // for loop
end // while loop
```

Outer loop is monotone increasing for ***FIRST*** sets ⇒ | T ∪ NT ∪ EOF ∪ ε | is bounded, so it terminates

Example

Consider the SheepNoise grammar and its ***FIRST*** sets

Goal	::= SheepNoise
SheepNoise	::= SheepNoise **baa** | **baa**

baa is a terminal symbol

Clearly, ***FIRST***(x) = {baa}, $\forall$ x $\in$ (**T** $\cup$ **NT**)

Symbol	***FIRST*** Set
Goal	**baa**
SheepNoise	**baa**
baa	**baa**

Computing *FIRST* sets

for each $x \in$ (T $\cup$ EOF $\cup$ ε)
    FIRST(x) ← {x}
for each A $\in$ **NT**, ***FIRST***(A) ← ∅

Initialization assigns each ***FIRST*** set a value

while (***FIRST*** sets are still changing) do
    for each p $\in$ P, of the form X → $Y_1 Y_2 \ldots Y_k$ do
    temp ← ***FIRST***(Y_1) - { ε }
        i ← 1
        while (i ≤ k-1 and ε $\in$ ***FIRST***(Y_i))
            temp ← temp $\cup$ (***FIRST***(Y_{i+1}) - { ε })
            i ← i + 1
        end // while loop
        if i == k and ε $\in$ ***FIRST***(Y_k)
            then temp ← temp $\cup$ { ε }
        FIRST(X) ← ***FIRST***(X) $\cup$ temp
      end // if - then
    end // for loop
end // while loop

Symbol	*FIRST* Set
Goal	
SheepNoise	
baa	

Computing *FIRST* sets

for each $x \in$ (T $\cup$ EOF $\cup$ ε)
    FIRST(x) $\leftarrow$ {x}
for each A $\in$ **NT**, ***FIRST***(A) $\leftarrow \varnothing$

while (***FIRST*** sets are still changing) do
    for each p $\in$ P, of the form X $\rightarrow$ $Y_1Y_2\ldots Y_k$ do
        $\boxed{\text{temp} \leftarrow \textit{FIRST}(Y_1) - \{\, \varepsilon \,\}}$
            i $\leftarrow$ 1
            while (i $\leq$ k-1 and $\varepsilon \in$ ***FIRST***(Y_i))
                temp $\leftarrow$ temp $\cup$ (***FIRST***(Y_{i+1}) - { ε })
                i $\leftarrow$ i + 1
            end // while loop
            if i == k and $\varepsilon \in$ ***FIRST***(Y_k)
                then temp $\leftarrow$ temp $\cup$ { ε }
            $\boxed{\textit{FIRST}(X) \leftarrow \textit{FIRST}(X) \cup \text{temp}}$
        end // if - then
    end // for loop
end // while loop

1	Goal	::= SheepNoise
2	SheepNoise	::= SheepNoise **baa**
3	SheepNoise	::= **baa**

If we visit the rule in the order 3, 2, 1

Symbol	***FIRST*** Set
Goal	$\varnothing$
SheepNoise	
baa	{baa}

Computing *FIRST* sets

```
for each x ∈ ( T ∪ EOF ∪ ε)
    FIRST(x) ← {x}
for each A ∈ NT, FIRST(A) ← ∅

while (FIRST sets are still changing) do
    for each p ∈ P, of the form X → Y1Y2…Yk do
        temp ← FIRST(Y1) - { ε }
        i ← 1
        while ( i ≤ k-1 and ε ∈ FIRST(Yi) )
            temp ← temp ∪ (FIRST(Yi+1) - { ε })
            i ← i + 1
        end // while loop
        if i == k and ε ∈ FIRST(Yk)
            then temp ← temp ∪ { ε }
        FIRST(X) ← FIRST(X) ∪ temp
      end // if - then
    end // for loop
end // while loop
```

1	Goal	::=	SheepNoise
2	SheepNoise	::=	SheepNoise **baa**
3	SheepNoise	::=	**baa**

If we visit the rule in the order 3, 2, 1

Symbol	*FIRST* Set
Goal	
SheepNoise	{baa}
baa	{baa}

An Example

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= LP List RP

Where LP is (and RP is)

If we visit the rules in order 4, 3, 2, 1 ⇒

Symbol	*Initial*	1st	2nd
Goal	∅		
List	∅		
Pair	∅		
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

An Example

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= <u>LP</u> List <u>RP</u>

Where <u>LP</u> is **(** and <u>RP</u> is **)**

If we visit the rules in order 4, 3, 2, 1 ⇒

Symbol	***Initial***	1st	2nd
Goal	∅		
List	∅		
Pair	∅	<u>LP</u>	
LP	<u>LP</u>	<u>LP</u>	<u>LP</u>
RP	<u>RP</u>	<u>RP</u>	<u>RP</u>
EOF	EOF	EOF	EOF

An Example

Consider the simplest parentheses grammar

1 Goal ::= List
2 List ::= Pair List
3 | ε
4 Pair ::= LP List RP

Where LP is (and RP is)

If we visit the rules in order 4, 3, 2, 1 ⇒

Symbol	*Initial*	1^{st}	2^{nd}
Goal	∅		
List	∅		
Pair	∅	LP	
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

An Example

Consider the simplest parentheses grammar

```
1  Goal ::= List
2  List ::= Pair List
3         |  ε
4  Pair ::= LP List RP
```

Where LP is (and RP is)

If we visit the rules in order 4, 3, 2, 1 $\Rightarrow$

Symbol	*Initial*	1st	2nd
Goal	∅		
List	∅	LP, ε	
Pair	∅	LP	
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

An Example

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= LP List RP

Where LP is **(** and RP is **)**

If we visit the rules in order 4, 3, 2, 1 ⇒

Symbol	***Initial***	1^{st}	2^{nd}
Goal	∅		
List	∅	LP, ε	
Pair	∅	LP	
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

An Example

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= LP List RP

Where LP is (and RP is)

If we visit the rules in order 4, 3, 2, 1 ⇒

Symbol	*Initial*	1^{st}	2^{nd}
Goal	∅	LP, ε	
List	∅	LP, ε	
Pair	∅	LP	
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

An Example

Consider the simplest parentheses grammar

1 Goal ::= List
2 List ::= Pair List
3 | ε
4 Pair ::= LP List RP

Where LP is (and RP is)

If we visit the rules in order 4, 3, 2, 1 ⇒

Symbol	*Initial*	1st	2nd
Goal	∅	LP, ε	
List	∅	LP, ε	
Pair	∅	LP	LP
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

An Example

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= LP List RP

Where LP is (and RP is)

If we visit the rules in order 4, 3, 2, 1 ⇒

Symbol	*Initial*	1st	2nd
Goal	∅	LP, ε	
List	∅	LP, ε	LP, ε
Pair	∅	LP	LP
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

An Example

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= LP List RP

Where LP is (and RP is)

If we visit the rules in order 4, 3, 2, 1 ⇒

Symbol	*Initial*	1st	2nd
Goal	∅	LP, ε	LP, ε
List	∅	LP, ε	LP, ε
Pair	∅	LP	LP
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

An Example

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= LP List RP

- Iteration 1 adds LP to ***FIRST***(Pair) and LP, ε to ***FIRST***(List) and ***FIRST***(Goal)

  ⇒ If we take them in rule order 4, 3, 2, 1
- Algorithm reaches fixed point at Iteration 2

Symbol	***Initial***	1^{st}	2^{nd}
Goal	∅	LP, ε	LP, ε
List	∅	LP, ε	LP, ε
Pair	∅	LP	LP
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

FOLLOW Sets

FOLLOW(A):

For A $\in$ **NT** , define **FOLLOW**(A) as the set of tokens that can occur immediately after A in a valid sentential form.

FOLLOW set is defined over the set of non-terminal symbols, **NT**.

Follow Set Construction

To Build FOLLOW(X) for non-terminal X:

- Place EOF in FOLLOW(<start>)
- 1. For each X as a non-terminal, initialize FOLLOW(X) to $\emptyset$

  2. ***Iterate until*** no more terminals can be added to any FOLLOW(X):

     For each rule *p* in the grammar

       If p is of the form A ::= $\alpha B \beta$, then

         if $\varepsilon \in FIRST(\beta)$

           Place {FIRST(β) - ε, FOLLOW(A)} in FOLLOW(B)

         else

           Place {FIRST(β)} in FOLLOW(B)

       If p is of the form A ::= αB, then

           Place FOLLOW(A) in FOLLOW(B)

  End iterate

Computing *FOLLOW* Sets

for each A ∈ **NT**
    FOLLOW(A) ← ∅
FOLLOW(S) ← { **EOF** }

while (***FOLLOW*** sets are still changing) do
    for each p ∈ P, of the form A → $B_1B_2\ldots B_k$ do
        TRAILER ← ***FOLLOW***(A)
        for i ← k down to 1
            if B_i ∈ **NT** then // domain checking
                FOLLOW(B_i) ← ***FOLLOW***(B_i) ∪ TRAILER
                if ε ∈ ***FIRST***(B_i) // add right context
                    TRAILER ← TRAILER ∪ (***FIRST***(B_i) - { ε })
                else TRAILER ← ***FIRST***(B_i) // no ε => truncate the right context
            else TRAILER ← { B_i } // B_i ∈ **T** => only 1 symbol

Don't add ε

To build ***FOLLOW*** sets, we need ***FIRST*** sets

Computing *FOLLOW* Sets

For a production $A \rightarrow B_1 B_2 \ldots B_k$:

- It works its way backward through the production: B_k, B_{k-1}, … B_1
- It builds the ***FOLLOW*** sets for the rhs symbols, B_1, B_2, … B_k, not A
- In the absence of ε, ***FOLLOW***(B_i) is just ***FIRST***(B_{i+1})
  - As always, ε makes the algorithm more complex

To handle ε, the algorithm keeps track of the first word in the trailing right context as it works its way back through rhs: B_k, B_{k-1}, … B_1

Computing *FOLLOW* Sets

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= <u>LP</u> List <u>RP</u>

Symbol	***Initial***
Goal	**EOF**
List	∅
Pair	∅

Initial Values:

- Goal, List and Pair are set to ∅
- Goal is then set to { **EOF** }

An Example

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= <u>LP</u> List <u>RP</u>

Symbol	*Initial*	1st
Goal	**EOF**	
List	∅	
Pair	∅	

Iteration 1:

If we visit the rules in order 1, 2, 3, 4

Symbol	*FIRST* Set
Goal	<u>LP</u>, ε
List	<u>LP</u>, ε
Pair	<u>LP</u>
LP	<u>LP</u>
RP	<u>RP</u>
EOF	EOF

Assume FIRST Sets are obtained using the algorithm we discussed in previous slides.

An Example

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= LP List RP

Symbol	*Initial*	1^{st}
Goal	**EOF**	**EOF**
List	∅	**EOF**
Pair	∅	

Iteration 1:

If we visit the rules in order 1, 2, 3, 4

Symbol	***FIRST*** Set
Goal	LP, ε
List	LP, ε
Pair	LP
LP	LP
RP	RP
EOF	EOF

An Example

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= LP List RP

Symbol	*Initial*	1^{st}
Goal	**EOF**	**EOF**
List	∅	**EOF**
Pair	∅	**EOF**, LP

Iteration 1:

If we visit the rules in order 1, 2, 3, 4

Symbol	*FIRST* Set
Goal	LP, ε
List	LP, ε
Pair	LP
LP	LP
RP	RP
EOF	EOF

An Example

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= <u>LP</u> List <u>RP</u>

Symbol	***Initial***	1^{st}
Goal	**EOF**	**EOF**
List	∅	**EOF**, RP
Pair	∅	**EOF**, LP

Iteration 1:

If we visit the rules in order 1, 2, 3, 4

Symbol	***FIRST*** Set
Goal	<u>LP</u>, ε
List	<u>LP</u>, ε
Pair	<u>LP</u>
LP	<u>LP</u>
RP	<u>RP</u>
EOF	EOF

An Example

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= <u>LP</u> List <u>RP</u>

Symbol	*Initial*	1^{st}
Goal	**EOF**	**EOF**
List	∅	**EOF**, RP
Pair	∅	**EOF**, LP

Iteration 1:

If we visit the rules in order 1, 2, 3, 4

Symbol	***FIRST*** Set
Goal	<u>LP</u>, ε
List	<u>LP</u>, ε
Pair	<u>LP</u>
LP	<u>LP</u>
RP	<u>RP</u>
EOF	EOF

An Example

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= LP List RP

Symbol	*Initial*	1st	2nd
Goal	**EOF**	**EOF**	
List	∅	**EOF**, RP	
Pair	∅	**EOF**, LP	

Iteration 2:

If we visit the rules in order 1, 2, 3, 4

Symbol	***FIRST*** Set
Goal	LP, ε
List	LP, ε
Pair	LP
LP	LP
RP	RP
EOF	EOF

An Example

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= LP List RP

Symbol	***Initial***	1^{st}	2^{nd}
Goal	**EOF**	**EOF**	**EOF**
List	∅	**EOF**, RP	**EOF**, RP
Pair	∅	**EOF**, LP	

Iteration 2:

If we visit the rules in order 1, 2, 3, 4

Symbol	***FIRST*** Set
Goal	LP, ε
List	LP, ε
Pair	LP
LP	LP
RP	RP
EOF	EOF

An Example

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= LP List RP

Symbol	*Initial*	1^{st}	2^{nd}
Goal	**EOF**	**EOF**	**EOF**
List	∅	**EOF**, RP	**EOF**, RP
Pair	∅	**EOF**, LP	**EOF**, RP, LP

Iteration 2:

Symbol	***FIRST*** Set
Goal	LP, ε
List	LP, ε
Pair	LP
LP	LP
RP	RP
EOF	EOF

If we visit the rules in order 1, 2, 3, 4

An Example

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= LP List RP

Iteration 2:

Symbol	***Initial***	1^{st}	2^{nd}
Goal	**EOF**	**EOF**	**EOF**
List	∅	**EOF**, RP	**EOF**, RP
Pair	∅	**EOF**, LP	**EOF**, RP, LP

If we visit the rules in order 1, 2, 3, 4

Symbol	***FIRST*** Set
Goal	LP, ε
List	LP, ε
Pair	LP
LP	LP
RP	RP
EOF	EOF

An Example

Consider the simplest parentheses grammar

1	Goal ::= List
2	List ::= Pair List
3	| ε
4	Pair ::= <u>LP</u> List <u>RP</u>

Symbol	*Initial*	1st	2nd
Goal	**EOF**	**EOF**	**EOF**
List	∅	**EOF**, RP	**EOF**, RP
Pair	∅	**EOF**, LP	**EOF**, RP, LP

Iteration 2:

- Production 1 adds nothing new
- Production 2 adds RP to ***FOLLOW***(Pair)

from ***FOLLOW***(List), ε ∈ ***FIRST***(List)

- Production 3 does nothing
- Production 4 adds nothing new

Symbol	***FIRST*** Set
Goal	<u>LP</u>, ε
List	<u>LP</u>, ε
Pair	<u>LP</u>
LP	<u>LP</u>
RP	<u>RP</u>
EOF	EOF

Iteration 3 produces the same result ⇒ reached a fixed point (omitted in the table)

Next Lecture

Things to do:

- Read Scott, Chapter 2.1 - 2.3.3; ALSU 2.4