CS 314 Principles of Programming Languages

Lecture 8: LL(1) Parsing

Prof. Zheng Zhang

Rutgers University

September 28, 2018

Class Information

- Homework 1 grades posted.
- Homework 3 posted, due Tuesday 10/02 11:55 pm EDT.

Review: FIRST and FOLLOW Sets

FIRST(α):

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

That is, $\mathbf{x} \in \text{FIRST}(\alpha)$ 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

First Set Example

Start ::= S **eof**
S ::= **a** S **b** | ε

$FIRST(\varepsilon) = \{\varepsilon\}$

S can be rewritten as the following:

ab
aaabbb
aabb
ε
…

$FIRST(S) = \{a, \varepsilon\}$

aSb can be rewritten as the following:

ab
aabb
…

$FIRST(aSb) = \{a\}$

Computing *FIRST* Sets

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

- 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 $B_1 B_2 \dots B_{m-1}$ 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 ::= ε, then ε ∈ FIRST(X)
- For each X as a non-terminal, initialize FIRST(X) to ∅
- ***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 $a \in \mathrm{FIRST}(Y_1)$

    add a to FIRST(X) if $a \in \mathrm{FIRST}(Y_i)$ and $\varepsilon \in \mathrm{FIRST}(Y_j)$

    for all $1 \leq j \leq i-1$ and $i \geq 2$

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

  EndFor

End iterate

Filling in the Details: Computing *FIRST* sets

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

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 $X \rightarrow Y_1 Y_2 \ldots Y_k$ 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

Filling in the Details: Computing *FIRST* sets

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

Initialize ***FIRST*** of each non-terminal symbol as empty set

while (***FIRST*** sets are still changing) do
    for each p ∈ P, of the form X → $Y_1Y_2\ldots Y_k$ 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

Filling in the Details: 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
```

If any ***FIRST*** set changes, it might affect other ***FIRST*** set(s) due to the inter-dependence relationship.

Filling in the Details: Computing *FIRST* sets

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

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

Check each rule in the grammar, see if any other ***FIRST*** set needs to be updated.

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 → $Y_1Y_2 \ldots Y_k$ 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

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 ∈ ( 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 all the rhs symbols can go to ε, then we add ε to ***FIRST***(lhs)

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 $X \rightarrow Y_1 Y_2 \ldots Y_k$ 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

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

An Example

Consider the simplest parentheses grammar

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

1st means first "while" iteration

Where LP is (and RP is)

```
……
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
```

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 ::= LP List RP

1st means first "while" iteration

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	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

Applying

Pair ::= LP List RP

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>

1st means first "while" iteration

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	∅	<u>LP</u>, ε	
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

Applying

List ::= Pair List
| ε

An Example

Consider the simplest parentheses grammar

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

1^{st} means first "while" iteration

Where LP is (and RP is)

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

Applying

Goal ::= List

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	$\mid$ ε
4	Pair ::= <u>LP</u> List <u>RP</u>

1st means first "while" iteration

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

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

Applying

Pair ::= <u>LP</u> List <u>RP</u>

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

1st means first "while" iteration

Where LP is (and RP is)

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

Applying

List ::= Pair List
| ε

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

1st means first "while" iteration

Where LP is (and RP is)

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

Applying

Goal ::= List

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 ::= <u>LP</u> List <u>RP</u>

1st means first "while" iteration

FIRST **Sets**

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

- 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

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**.

Back to Our Example

Start ::= S **eof**
S ::= **a** S **b** |
ε

One possible derivation process from the start symbol:

Start ⇒

FOLLOW(S) = { }

Back to Our Example

Start ::= S **eof**
S ::= **a** S **b** |
ε

One possible derivation process from the start symbol:

Start $\Rightarrow$ S **eof**

FOLLOW(S) = { }

Back to Our Example

Start ::= S **eof**
S ::= **a** S **b** | ε

One possible derivation process from the start symbol:

Start ⇒ S **eof** ⇒ **a** S **b** **eof**

FOLLOW(S) = { }

Back to Our Example

Start ::= S **eof**
S ::= **a** S **b** |
ε

One possible derivation process from the start symbol:

Start ⇒ S **eof** ⇒ **a** S **b eof** ⇒ **a b eof**

FOLLOW(S) = { }

Back to Our Example

Start ::= S **eof**
S ::= **a** S **b** |
ε

One possible derivation process from the start symbol:

Start ⇒ S **eof** ⇒ **a** S **b eof** ⇒ **a b eof**

FOLLOW(S) = { eof , b }

Computing *FOLLOW* Sets

For a production $A \rightarrow B_1 B_2 \ldots B_{k-1} B_k$:

- FOLLOW(B_k) includes FOLLOW(A)
- FOLLOW(B_{k-1}) includes
  FIRST(B_k) - ε,
  and FOLLOW(A) if B_k can derive ε
- FOLLOW(B_{k-2}) includes
  FIRST($B_{k-1}B_k$) - ε,
  and FOLLOW(A) if $B_{k-1}B_k$ can derive ε
- …

Follow Set Construction

Given a rule *p* in the grammar:

$$A \rightarrow B_1B_2\ldots B_iB_{i+1}\ldots B_k$$

If B_i is a non-terminal, FOLLOW(B_i) includes

- FIRST($B_{i+1}\ldots B_k$) - {ε} U FOLLOW(A), if $\varepsilon \in$ FIRST($B_{i+1}\ldots B_k$)
- FIRST($B_{i+1}\ldots B_k$) otherwise

Follow Set Construction

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

- Place EOF in FOLLOW(<start>)
- For each X as a non-terminal, initialize FOLLOW(X) to ∅

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 ::= αBβ, then

if ε ∈ *FIRST*(β)

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 → B1B2…Bk do
        TRAILER ← FOLLOW(A)
        for i ← k down to 1
            if Bi ∈ NT then
                FOLLOW(Bi) ← FOLLOW(Bi) ∪ TRAILER
                if ε ∈ FIRST(Bi)
                    TRAILER ← TRAILER ∪ (FIRST(Bi) - { ε })
                else TRAILER ← FIRST(Bi)
            else TRAILER ← { Bi }
```

Initially, set ***FOLLOW*** for each non-terminal symbol

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

Computing *FOLLOW* Sets

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

Set ***FOLLOW*** for start symbol S as {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
                FOLLOW(B_i) ← ***FOLLOW***(B_i) ∪ TRAILER
                if ε ∈ ***FIRST***(B_i)
                    TRAILER ← TRAILER ∪ (***FIRST***(B_i) - { ε })
                else TRAILER ← ***FIRST***(B_i)
            else TRAILER ← { B_i }

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

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 → B1B2…Bk do
        TRAILER ← FOLLOW(A)
        for i ← k down to 1
            if Bi ∈ NT then
                FOLLOW(Bi) ← FOLLOW(Bi) ∪ TRAILER
                if ε ∈ FIRST(Bi)
                    TRAILER ← TRAILER ∪ (FIRST(Bi) - { ε })
                else TRAILER ← FIRST(Bi)
            else TRAILER ← { Bi }
```

As long as any ***FOLLOW*** set changes

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

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
                FOLLOW(B_i) ← ***FOLLOW***(B_i) ∪ TRAILER
                if ε ∈ ***FIRST***(B_i)
                    TRAILER ← TRAILER ∪ (***FIRST***(B_i) - { ε })
                else TRAILER ← ***FIRST***(B_i)
            else TRAILER ← { B_i }

As long as any ***FOLLOW*** set changes, check all the rules

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

Computing *FOLLOW* Sets

for each $A \in$ **NT**
    FOLLOW(A) $\leftarrow \varnothing$
FOLLOW(S) $\leftarrow$ { **EOF** }

while (***FOLLOW*** sets are still changing) do
    for each $p \in P$, of the form $A \rightarrow B_1B_2\ldots B_k$ do
        TRAILER $\leftarrow$ ***FOLLOW***(A)
        for $i \leftarrow k$ down to 1
            if $B_i \in$ **NT** then
                FOLLOW(B_i) $\leftarrow$ ***FOLLOW***(B_i) $\cup$ TRAILER
                if $\varepsilon \in$ ***FIRST***(B_i)
                    TRAILER $\leftarrow$ TRAILER $\cup$ (***FIRST***(B_i) - { ε })
                else TRAILER $\leftarrow$ ***FIRST***(B_i)
            else TRAILER $\leftarrow$ { B_i }

set trailing context to ***FOLLOW***(A)

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

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
        FOLLOW(B_i) ← ***FOLLOW***(B_i) ∪ TRAILER
        if ε ∈ ***FIRST***(B_i)
          TRAILER ← TRAILER ∪ (***FIRST***(B_i) - { ε })
        else TRAILER ← ***FIRST***(B_i)
      else TRAILER ← { B_i }

it goes backwards

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

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 $A \rightarrow B_1B_2\ldots B_k$ do
        TRAILER ← ***FOLLOW***(A)
        for i ← k down to 1
            if B_i ∈ **NT** then
                FOLLOW(B_i) ← ***FOLLOW***(B_i) ∪ TRAILER
                if ε ∈ ***FIRST***(B_i)
                    TRAILER ← TRAILER ∪ (***FIRST***(B_i) - { ε })
                else TRAILER ← ***FIRST***(B_i)
            else TRAILER ← { B_i }

if the symbol is non-terminal, need to check if it derives ε

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

Computing *FOLLOW* Sets

for each $A \in$ **NT**
    FOLLOW(A) $\leftarrow \varnothing$
FOLLOW(S) $\leftarrow$ { **EOF** }

while (***FOLLOW*** sets are still changing) do
    for each $p \in P$, of the form $A \rightarrow B_1 B_2 \ldots B_k$ do
        TRAILER $\leftarrow$ ***FOLLOW***(A)
        for $i \leftarrow k$ down to 1
            if $B_i \in$ **NT** then
                FOLLOW(B_i) $\leftarrow$ ***FOLLOW***(B_i) $\cup$ TRAILER
                if $\varepsilon \in$ ***FIRST***(B_i)
                    TRAILER $\leftarrow$ TRAILER $\cup$ (***FIRST***(B_i) - { ε })
                else TRAILER $\leftarrow$ ***FIRST***(B_i)
            else TRAILER $\leftarrow$ { B_i }

Consecutive non-terminals that derive ε in trailing context

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

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 → $B_1 B_2 \ldots B_k$ do
    TRAILER ← ***FOLLOW***(A)
    for i ← k down to 1
      if B_i ∈ **NT** then
        FOLLOW(B_i) ← ***FOLLOW***(B_i) ∪ TRAILER
        if ε ∈ ***FIRST***(B_i)
          TRAILER ← TRAILER ∪ (***FIRST***(B_i) - { ε })
        else TRAILER ← ***FIRST***(B_i)
      else TRAILER ← { B_i }

Trailing context needs to be reset

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

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 $A \rightarrow B_1 B_2 \ldots B_k$ do
        TRAILER ← ***FOLLOW***(A)
        for i ← k down to 1
            if B_i ∈ **NT** then
                FOLLOW(B_i) ← ***FOLLOW***(B_i) ∪ TRAILER
                if ε ∈ ***FIRST***(B_i)
                    TRAILER ← TRAILER ∪ (***FIRST***(B_i) - { ε })
                else TRAILER ← ***FIRST***(B_i)
            else TRAILER ← { B_i }

when B_i does not derive ε

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

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***
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 ::= LP List RP

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

Iteration 1 (of while loop):

```
while (FOLLOW sets are still changing) do
    for each p ∈ P, of the form A → B1B2…Bk do
     TRAILER ← FOLLOW(A)
     for i ← k down to 1
         if Bi ∈ NT then
             FOLLOW(Bi) ← FOLLOW(Bi) ∪ TRAILER
        if ε ∈ FIRST(Bi)
                TRAILER ← TRAILER ∪ (FIRST(Bi) - { ε })
        else TRAILER ← FIRST(Bi)
       else TRAILER ← { Bi }
```

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	∅	
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	∅	

Iteration 1:

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

Goal ::= List

Add FOLLOW(Goal) to FOLLOW(List)

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	∅	

Iteration 1:

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

List ::= Pair List

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:

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

List ::= Pair List

- Add FIRST(List) to FOLLOW(Pair)
- Add FOLLOW(List) to FOLLOW(Pair)

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

Pair ::= LP List RP

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
Goal	**EOF**	**EOF**
List	∅	**EOF**, RP
Pair	∅	**EOF**, LP

Iteration 1:

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

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

Pair ::= <u>LP</u> List <u>RP</u>

- Add FIRST(<u>RP</u>) to FOLLOW(List)

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**, RP
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}	2^{nd}
Goal	**EOF**	**EOF**	**EOF**
List	∅	**EOF**, RP	**EOF**, RP
Pair	∅	**EOF**, LP	**EOF**, LP

Iteration 2:

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

Goal ::= List

~~Add FOLLOW(Goal) to FOLLOW(List)~~

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**, LP, RP

Iteration 2:

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

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

List ::= Pair List

- ~~Add FIRST(List) to FOLLOW(Pair)~~
- Add FOLLOW(List) to FOLLOW(Pair)

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

Pair ::= LP List RP

- ~~Add FIRST(RP) to FOLLOW(List)~~

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:

- 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	LP, ε
List	LP, ε
Pair	LP
LP	LP
RP	RP
EOF	EOF

Iteration 3 produces the same result ⇒ reached a fixed point

Review: LL(1) Predictive Parsing

Key Property:

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

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

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) = ∅

Building the PREDICT set

- Need a ***PREDICT set*** for every rule

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

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

Symbol	*FIRST*	*FOLLOW*
Goal	LP, ε	**EOF**
List	LP, ε	**EOF**, RP
Pair	LP	**EOF**, RP, LP
LP	LP	-
RP	RP	-
EOF	EOF	-

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

Rule	*PREDICT*
1	**EOF, LP**
2	**LP**
3	**EOF**, RP
4	LP

Building the PREDICT set

- Need a ***PREDICT set*** for every rule

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

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

Symbol	*FIRST*	*FOLLOW*
Goal	LP, ε	**EOF**
List	LP, ε	**EOF**, RP
Pair	LP	**EOF**, RP, LP
LP	LP	-
RP	RP	-
EOF	EOF	-

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

Rule	*PREDICT*	
1	**EOF, LP**	
2	**LP**	← FIRST(Pair List)
3	**EOF**, RP	← FOLLOW(List)
4	LP	

Building the complete parse table

- Need a row for every **NT** and a column for every **T**
- Need an interpreter for the table (skeleton parser)

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

Rule	*PREDICT*
1	**EOF, LP**
2	**LP**
3	**EOF**, RP
4	LP

	LP	*RP*	*EOF*
Goal	**1**		**1**
List			
Pair			

Building the complete parse table

- Need a row for every **NT** and a column for every **T**
- Need an interpreter for the table (skeleton parser)

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

Rule	*PREDICT*
1	**EOF, RP**
2	**LP**
3	**EOF**, RP
4	LP

	LP	*RP*	*EOF*
Goal	**1**		**1**
List	**2**	**3**	**3**
Pair			

Building the complete parse table

- Need a row for every **NT** and a column for every **T**
- Need an interpreter for the table (skeleton parser)

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

Rule	*PREDICT*
1	**EOF, RP**
2	**LP**
3	**EOF**, RP
4	LP

	LP	*RP*	*EOF*
Goal	**1**		**1**
List	2	3	3
Pair	**4**		

Building the complete parse table

- Need a row for every **NT** and a column for every **T**
- Need an interpreter for the table (skeleton parser)

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

Rule	*PREDICT*
1	**EOF, RP**
2	**LP**
3	**EOF**, RP
4	LP

	LP	*RP*	*EOF*
Goal	**1**		**1**
List	2	3	3
Pair	4		

Next Lecture

Things to do:

- Start programming in C.
- Read Scott, Chapter 3.1 - 3.3; ALSU 7.1
- Read Scott, Chapter 8.1 - 8.2; ALSU 7.1 - 7.3