CS 314 Principles of Programming Languages

Lecture 8: LL(1) Parsing

Zheng (Eddy) Zhang

Rutgers University

February 12, 2018

Class Information

- Homework 3 posted, due Sunday 2/18 11:55 pm EST.

Review: LL(1) Predictive Parsing

Key Property:

Whenever two productions $A ::= \alpha$ and $A ::= \beta$ 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!

FIRST and FOLLOW Sets

FIRST(α):

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

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

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

T: terminals NT: non-terminals

First Set Example

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

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

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

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

FIRST and 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

FOLLOW(S) = { eof , b }

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 $a \in FIRST(Y_1)$

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

            for all $1 \le j \le i-1$ and $i \ge 2$

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

    EndFor

End iterate

Filling in the Details: Computing *FIRST* sets

$\boxed{\begin{array}{l}\text{for each } x \in (\text{ T} \cup \text{EOF} \cup \varepsilon) \\ \quad \textbf{\textit{FIRST}}(x) \leftarrow \{x\}\end{array}}$

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

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

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

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

Computing *FIRST* sets

for each $x \in$ (T $\cup$ EOF $\cup$ 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

Example

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

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

baa is a terminal symbol

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

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

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

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 $\rightarrow$ $Y_1 Y_2 \ldots Y_k$ do
    temp $\leftarrow$ ***FIRST***(Y_1) - { ε }
        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 }
        FIRST(X) $\leftarrow$ ***FIRST***(X) $\cup$ 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 $x \in$ (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

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*	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	∅	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*	1^{st}	2^{nd}
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 ::= <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 $\Rightarrow$

Symbol	*Initial*	1^{st}	2^{nd}
Goal	$\emptyset$	<u>LP</u>, ε	
List	$\emptyset$	<u>LP</u>, ε	<u>LP</u>, ε
Pair	$\emptyset$	<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

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

- 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

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

FIRST and 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>)
- 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** }

Don't add ε

```
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                          // domain checking
                FOLLOW(Bi) ← FOLLOW(Bi) ∪ TRAILER
                if ε ∈ FIRST(Bi)                     // add right context
                    TRAILER ← TRAILER ∪ (FIRST(Bi) - { ε })
                else TRAILER ← FIRST(Bi)  // no ε => truncate the right context
            else TRAILER ← { Bi }                   // Bi ∈ T => only 1 symbol
```

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: $B_k, B_{k-1}, \ldots B_1$
- It builds the ***FOLLOW*** sets for the rhs symbols, $B_1, B_2, \ldots B_k$, 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: $B_k, B_{k-1}, \ldots B_1$

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**	
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

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***	1^{st}	2^{nd}
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*	1st	2nd
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

Iteration 2:

Symbol	*Initial*	1st	2nd
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 ::= 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***	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	<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

Building the *FIRST*+ set

- Need a ***FIRST+ set*** for every rule

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	| ε
4	Pair ::= LP List RP

Rule	*FIRST+*
1	**EOF, LP**
2	**LP**
3	**EOF**, RP
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	| ε
4	Pair ::= LP List RP

Rule	*FIRST+*
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	| ε
4	Pair ::= <u>LP</u> List <u>RP</u>

Rule	*FIRST+*
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 ::= LP List RP

Rule	*FIRST+*
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	*FIRST+*
1	**EOF, RP**
2	**LP**
3	**EOF**, RP
4	LP

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

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()

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

M is the parse table

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