CS 314 Principles of Programming Languages

Lecture 9: LL(1) Parsing Review

Prof. Zheng Zhang

Rutgers University

October 3, 2018

Class Information

- Homework 4 will be posted after lecture 10.
- Project 1 will be posted after homework 4 is due.

Review: FIRST and FOLLOW Sets

FIRST(α):

For some α ∈ (T ∪ NT ∪ EOF ∪ 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** ∈ FIRST(α) iff α ⇒* **x**γ for some γ

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
…

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

aSb can be rewritten as the following:

ab
aabb
…

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

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) excluding 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 $a \in FIRST(Y_1)$
  add a to FIRST(X) if $a \in FIRST(Y_i)$ and $\varepsilon \in FIRST(Y_j)$
  for all $1 \leq j \leq i\text{-}1$ and $i \geq 2$
  add ε to FIRST(X) if $\varepsilon \in FIRST(Y_i)$ for all $1 \leq i \leq k$
  EndFor

End iterate

An Example

parentheses grammar

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

Iter. 1 means iteration 1

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

Initialization

- 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

Symbol	***Initial***	Iter. 1	Iter. 2
Goal	∅		
List	∅		
Pair	∅		
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

An Example

parentheses grammar

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

Iter. 1 means iteration 1

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

Iteration 1 (of the outer loop below):

- 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

Symbol	***Initial***	Iter. 1	Iter. 2
Goal	∅		
List	∅		
Pair	∅		
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

An Example

parentheses grammar

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

Iter. 1 means iteration 1

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

Iteration 1:

The order of the rules do not affect the final FIRST set results:

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

FIRST sets in progress

Symbol	***Initial***	Iter. 1	Iter. 2
Goal	∅		
List	∅		
Pair	∅		
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

An Example

parentheses grammar

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

Iter. 1 means iteration 1

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

Iteration 1:

FIRST sets in progress

Symbol	***Initial***	Iter. 1	Iter. 2
Goal	∅		
List	∅		
Pair	∅	?	
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

Applying Rule 4

Pair ::= LP List RP

add first(LP list RP) to first(Pair)

An Example

parentheses grammar

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

Iter. 1 means iteration 1

Where LP is (and RP is)

Iteration 1:

FIRST sets in progress

Symbol	***Initial***	Iter. 1	Iter. 2
Goal	∅		
List	∅		
Pair	∅	LP	
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

Applying Rule 4

Pair ::= LP List RP

add first(LP list RP) to first(Pair)

An Example

parentheses grammar

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

Iter. 1 means iteration 1

Where LP is (and RP is)

Iteration 1:

FIRST sets in progress

Symbol	***Initial***	Iter. 1	Iter. 2
Goal	∅		
List	∅		
Pair	∅	LP	
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

Applying Rule 4

Pair ::= LP List RP

add first(LP list RP) to first(Pair)

An Example

parentheses grammar

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

Iter. 1 means iteration 1

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

Iteration 1:

FIRST sets in progress

Symbol	***Initial***	Iter. 1	Iter. 2
Goal	∅		
List	∅	?	
Pair	∅	LP	
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

Applying Rule 2 and Rule 3

List ::= Pair List

| ε

add first(Pair List) to first(List)

add first(ε) to first(List)

An Example

parentheses grammar

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

Iter. 1 means iteration 1

Where LP is (and RP is)

FIRST sets in progress

Iteration 1:

Symbol	***Initial***	Iter. 1	Iter. 2
Goal	∅		
List	∅	LP, ε	
Pair	∅	LP	
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

Applying Rule 2 and Rule 3

List ::= Pair List

| ε

add first(Pair List) to first(List)

add first(ε) to first(List)

An Example

parentheses grammar

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

Iter. 1 means iteration 1

Where LP is (and RP is)

FIRST sets in progress

Iteration 1:

Symbol	***Initial***	Iter. 1	Iter. 2
Goal	∅		
List	∅	LP, ε	
Pair	∅	LP	
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

Applying Rule 2 and Rule 3

List ::= Pair List

| ε

add first(Pair List) to first(List)

add first(ε) to first(List)

An Example

parentheses grammar

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

Iter. 1 means iteration 1

Where LP is (and RP is)

Iteration 1:

Applying Rule 1

Goal ::= List

add first(List) to first(Goal)

FIRST sets in progress

Symbol	***Initial***	Iter. 1	Iter. 2
Goal	∅	?	
List	∅	LP, ε	
Pair	∅	LP	
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

An Example

parentheses grammar

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

Iter. 1 means iteration 1

Where LP is (and RP is)

Iteration 1:

FIRST sets in progress

Symbol	*Initial*	Iter. 1	Iter. 2
Goal	∅	LP, ε	
List	∅	LP, ε	
Pair	∅	LP	
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

Applying Rule 1

Goal ::= List

add first(List) to first(Goal)

An Example

parentheses grammar

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

Iter. 1 means iteration 1

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

FIRST sets in progress

Symbol	***Initial***	Iter. 1	Iter. 2
Goal	∅	LP, ε	
List	∅	LP, ε	
Pair	∅	LP	
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

We just finished the first iteration!
Recall that one iteration reviews all the rules!

An Example

parentheses grammar

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

Iter. 1 means iteration 1

Where LP is (and RP is)

FIRST sets in progress

Symbol	***Initial***	Iter. 1	Iter. 2
Goal	∅	LP, ε	
List	∅	LP, ε	
Pair	∅	LP	
LP	LP	LP	LP
RP	RP	RP	RP
EOF	EOF	EOF	EOF

Before the second iteration starts…

An Example

parentheses grammar

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

Iter. 1 means iteration 1

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

FIRST sets in progress

Symbol	***Initial***	Iter. 1	Iter. 2
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

Before the second iteration starts…

An Example

parentheses grammar

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

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

Iteration 2:

FIRST sets in progress

Symbol	***Initial***	Iter. 1	Iter. 2
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

Applying Rule 4

Pair ::= LP List RP

add first(LP list RP) to first(Pair)

LP is already in first(Pair)

An Example

parentheses grammar

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

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

Iteration 2:

FIRST sets in progress

Symbol	***Initial***	Iter. 1	Iter. 2
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

Applying Rule 2 and Rule 3

List ::= Pair List

| ε

add first(Pair List) to first(List)

add first(ε) to first(List)

LP and ε are already in FIRST(List)

An Example

parentheses grammar

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

Where LP is (and RP is)

Iteration 2:

FIRST sets in progress

Symbol	***Initial***	Iter. 1	Iter. 2
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

Applying Rule 1

Goal ::= List

add first(List) to first(Goal)

LP and ε are already in FIRST(Goal)

An Example

parentheses grammar

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

Comparing the FIRST sets at the end of iteration 1 and the end of iteration 2, nothing new is added.

↓

Reached fixed point! We have constructed complete FIRST sets!

FIRST Sets

Symbol	***Initial***	Iter. 1	Iter. 2
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**.

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 }

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

Given a rule *p* in the grammar:

$$A \rightarrow B_1 B_2 \ldots B_i B_{i+1} \ldots B_k$$

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

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

Relationship between FOLLOW sets and FIRST sets of different symbols

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

An Example for FOLLOW Set Construction

parentheses grammar

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

FOLLOW sets in progress

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

Initialization

- 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

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

An Example for FOLLOW Set Construction

parentheses grammar

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

FOLLOW sets in progress

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

Iteration 1 (of the outer loop below):

- 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

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

An Example for FOLLOW Set Construction

parentheses grammar

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

FOLLOW sets in progress

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

Iteration 1:

The order of the rules do not affect the final FOLLOW set results:

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 for FOLLOW Set Construction

parentheses grammar

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

FOLLOW sets in progress

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

Iteration 1:

Rule 1 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 for FOLLOW Set Construction

parentheses grammar

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

FOLLOW sets in progress

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

Iteration 1:

Rule 1 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 for FOLLOW Set Construction

parentheses grammar

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

FOLLOW sets in progress

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

Iteration 1:

Rule 2 List ::= Pair List

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

An Example for FOLLOW Set Construction

parentheses grammar

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

FOLLOW sets in progress

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

Iteration 1:

Rule 2 List ::= Pair List

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

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

An Example for FOLLOW Set Construction

parentheses grammar

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

FOLLOW sets in progress

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

Iteration 1:

Rule 2 List ::= Pair List

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

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

An Example for FOLLOW Set Construction

parentheses grammar

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

FOLLOW sets in progress

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

Iteration 1:

Rule 4 Pair ::= LP List RP

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

An Example for FOLLOW Set Construction

parentheses grammar

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

FOLLOW sets in progress

Symbol	*Initial*	1st
Goal	**EOF**	**EOF**
List	∅	**EOF**, ?
Pair	∅	**EOF**, LP

Iteration 1:

Rule 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 for FOLLOW Set Construction

parentheses grammar

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

FOLLOW sets in progress

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

Iteration 1:

Rule 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 for FOLLOW Set Construction

parentheses grammar

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

FOLLOW sets in progress

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

End of First Iteration and Before the Second Iteration starts

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

An Example for FOLLOW Set Construction

parentheses grammar

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

FOLLOW sets in progress

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

End of First Iteration and Before the Second Iteration starts

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

An Example for FOLLOW Set Construction

parentheses grammar

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

FOLLOW sets in progress

Symbol	*Initial*	1st	2nd
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

Rule 1 Goal ::= List

- Add FOLLOW(Goal) to FOLLOW(List)

EOF already in FOLLOW(list)

An Example for FOLLOW Set Construction

parentheses grammar

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

FOLLOW sets in progress

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

Rule 2 List ::= Pair List

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

→ Added RP

An Example for FOLLOW Set Construction

parentheses grammar

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

FOLLOW sets in progress

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

Iteration 2:

Rule 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

RP already in FOLLOW(list)

An Example for FOLLOW Set Construction

parentheses grammar

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

FOLLOW sets in progress

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

Iteration 2:

Iteration 3 produces the same result
⇒ reached a fixed point

We omit the results of Iteration 3.

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

Building Top-down Parsers

Building the PREDICT set

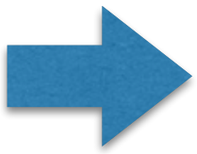
- Need a ***PREDICT set*** for every rule

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 Top-down Parsers

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	-

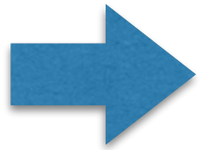
1	Goal ::= List
2	List ::= Pair List
3	List ::= ε
4	Pair ::= LP List RP

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

Building Top-down Parsers

Parentheses grammar

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



PREDICT Sets

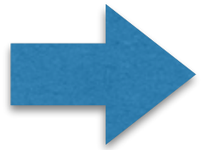
Rule	*PREDICT*
1	**EOF, LP**
2	**LP**
3	**EOF**, RP
4	LP

Is this grammar LL(1)?

Building Top-down Parsers

Parentheses grammar

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



PREDICT Sets

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

Is this grammar LL(1)?

Since only Rule 2 and Rule 3 correspond to the same non-terminal, and PREDICT(Rule 2) and PREDICT(Rule 3) are disjoint, the grammar is LL(1).

Building Top-down Parsers

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			
List			
Pair			

Building Top-down Parsers

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 Top-down Parsers

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 Top-down Parsers

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		

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 → Y1Y2 . . . Yk then
              pop X
              push Yk, Yk–1, . . . , Y1
          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		

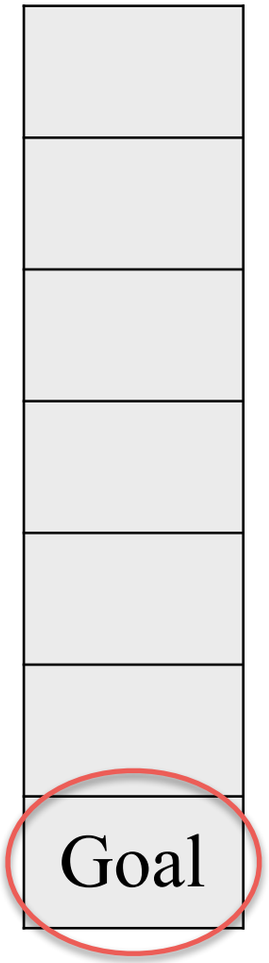
M is the parse table

LL(1) Parsing Example

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

Goal

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



Remaining Input:
LP RP LP LP RP RP

Sentential Form:
Goal

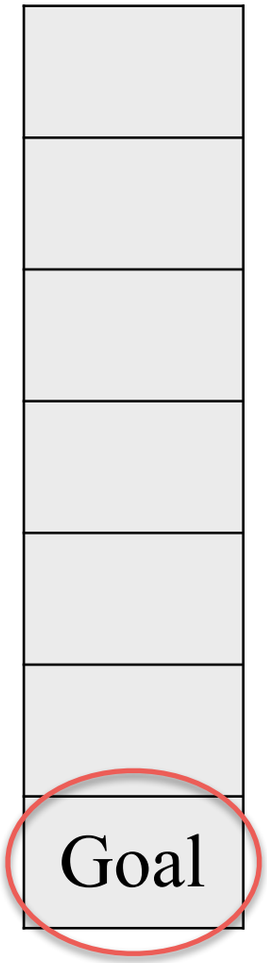
Applied Production:

LL(1) Parsing Example

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

Goal

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



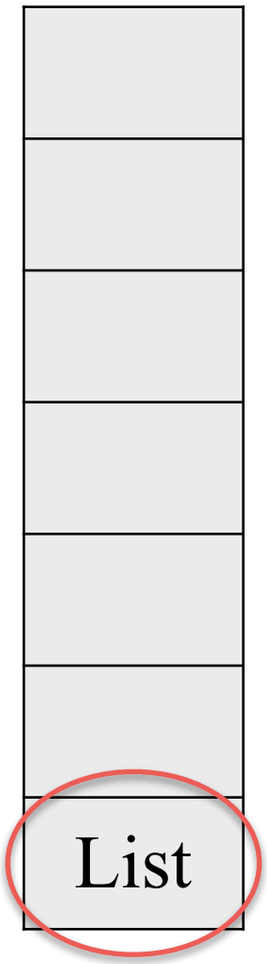
Remaining Input:
LP RP LP LP RP RP

Sentential Form:
Goal

Applied Production:

LL(1) Parsing Example

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



Goal

↓

List

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

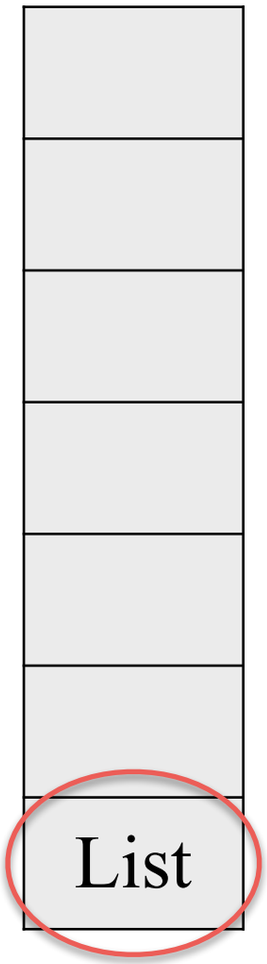
Remaining Input:
LP RP LP LP RP RP

Sentential Form:
List

Applied Production:
1. Goal ::= List

LL(1) Parsing Example

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



Goal

↓

List

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

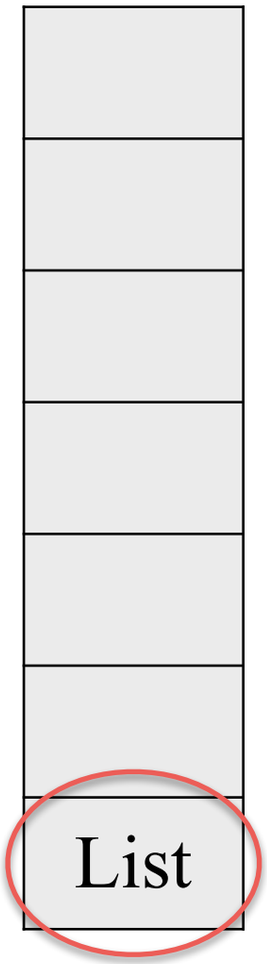
Remaining Input:
LP RP LP LP RP RP

Sentential Form:
List

Applied Production:
1. Goal ::= List

LL(1) Parsing Example

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



Goal

↓

List

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

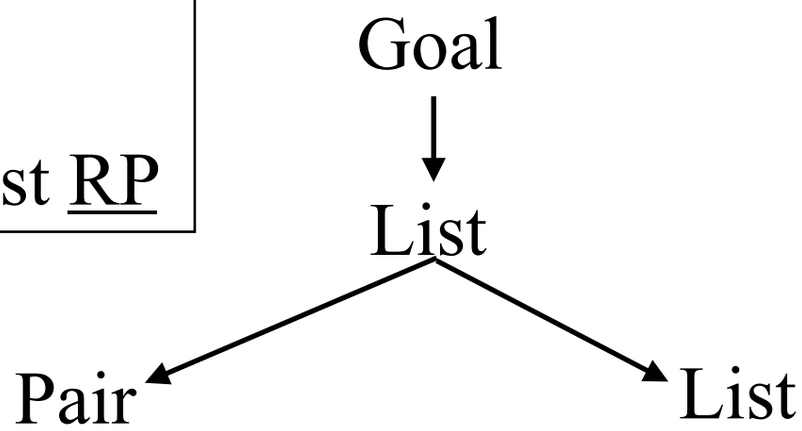
Remaining Input:
LP RP LP LP RP RP

Sentential Form:
List

Applied Production:
1. Goal ::= List

LL(1) Parsing Example

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



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

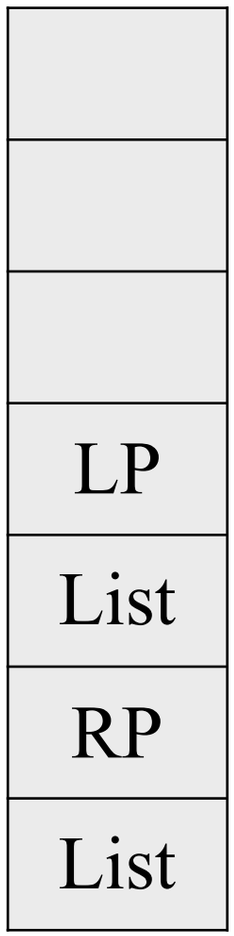
Remaining Input:
LP RP LP LP RP RP

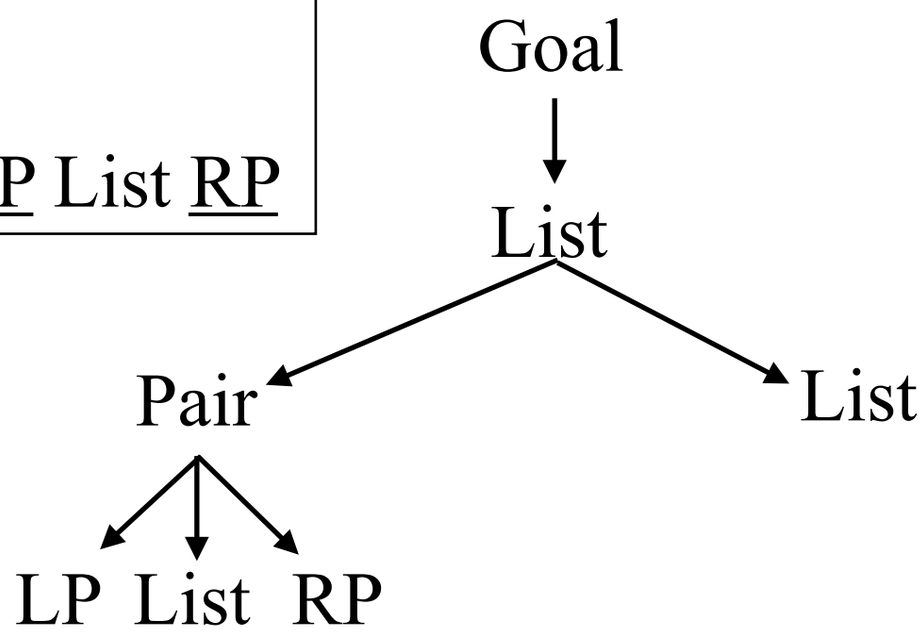
Sentential Form:
Pair List

Applied Production:
2. List ::= Pair List

LL(1) Parsing Example

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





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

Remaining Input:
LP RP LP LP RP RP

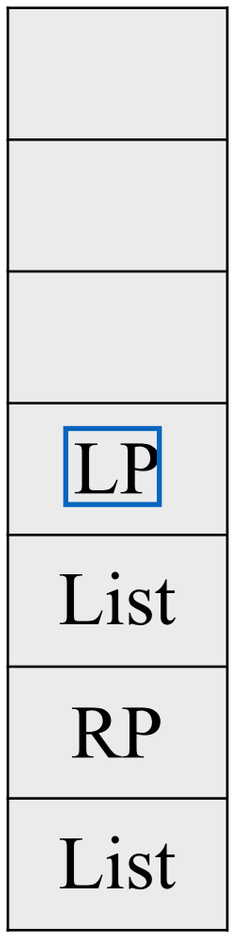
Sentential Form:
LP List RP List

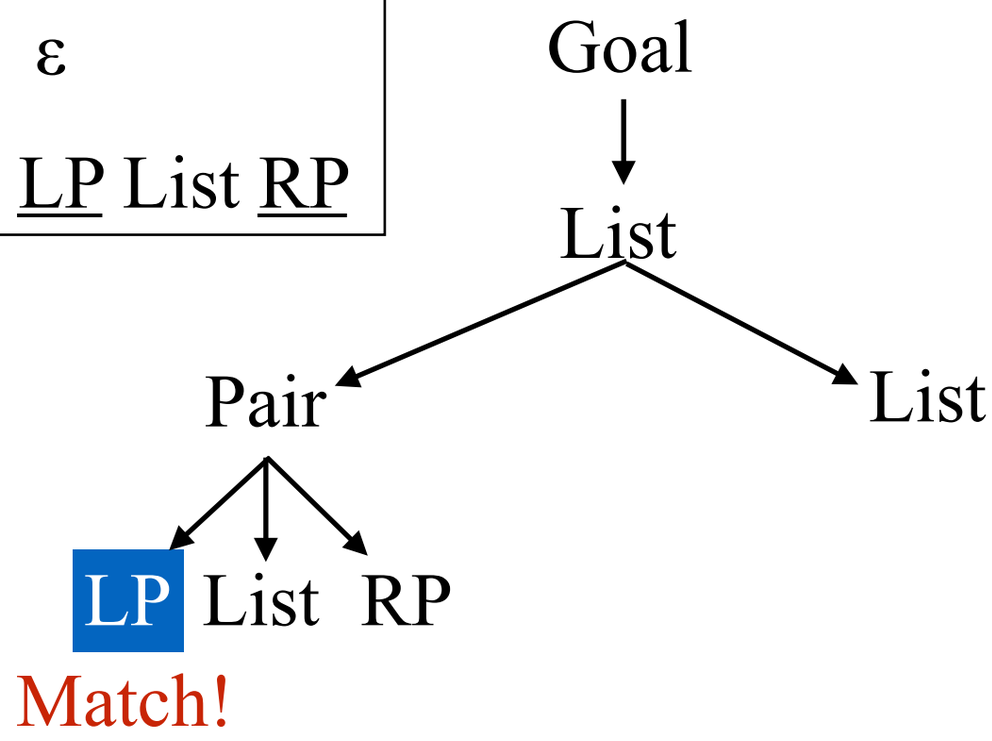
Applied Production:
4. Pair ::= LP List RP

LL(1) Parsing Example

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

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





Remaining Input:
LP RP LP LP RP RP

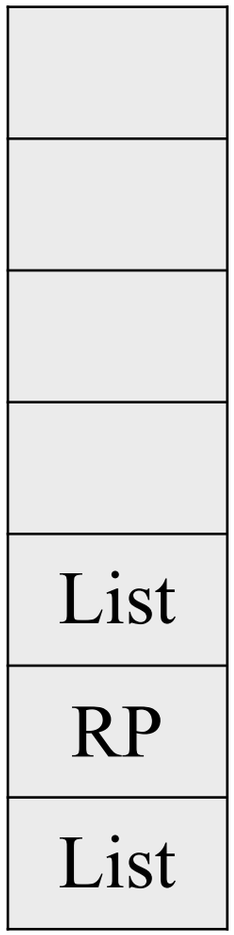
Sentential Form:
LP List RP List

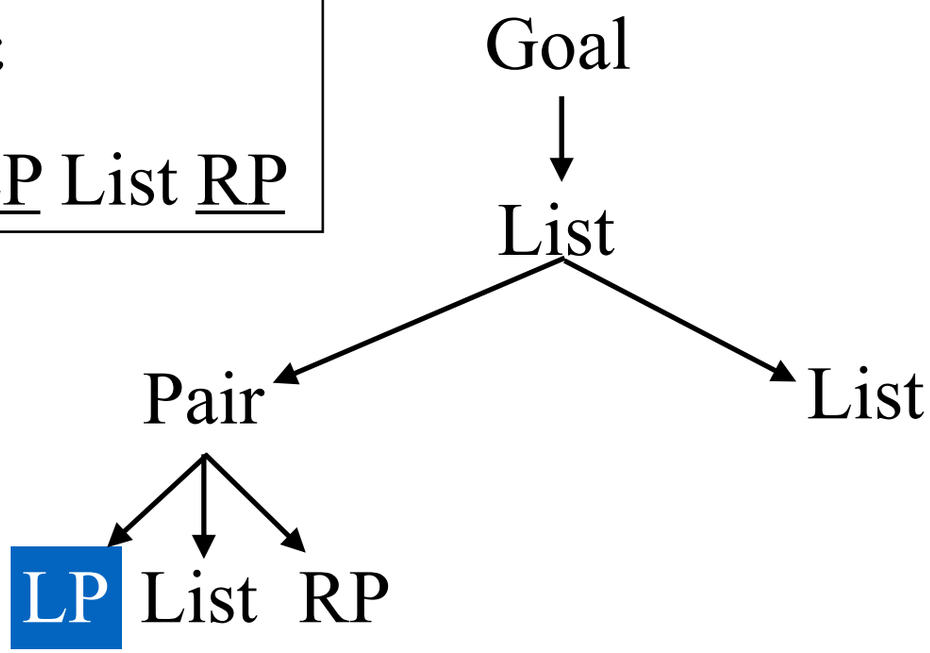
Applied Production:

LL(1) Parsing Example

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

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





Remaining Input:
RP LP LP RP RP

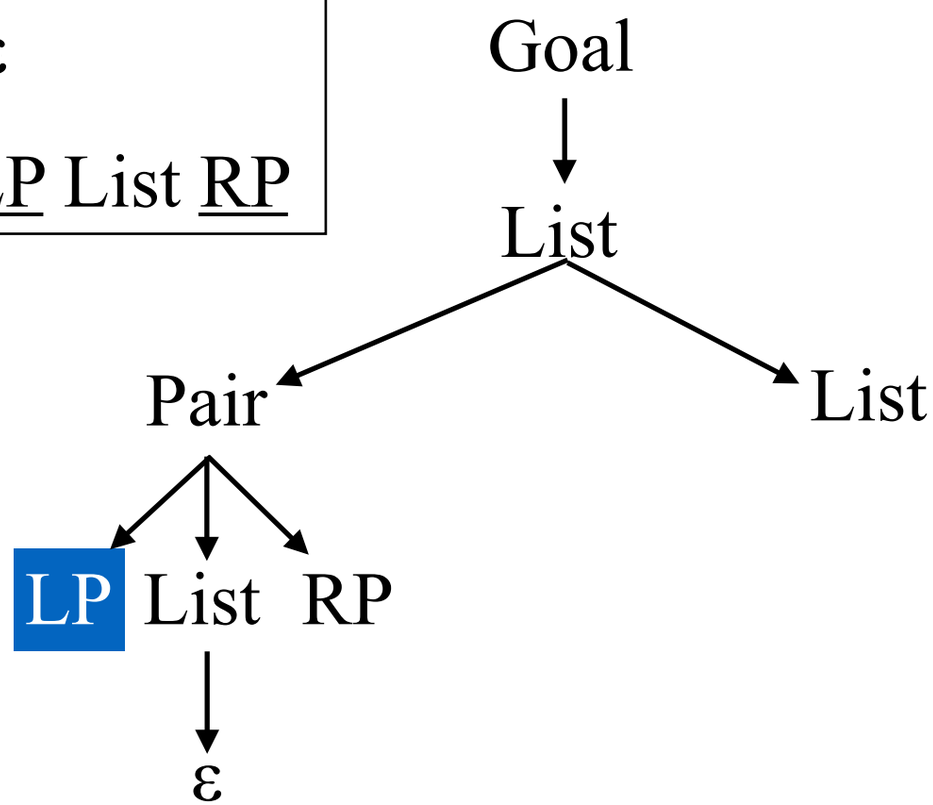
Sentential Form:
LP List RP List

Applied Production:

LL(1) Parsing Example

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

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



Remaining Input:
RP LP LP RP RP

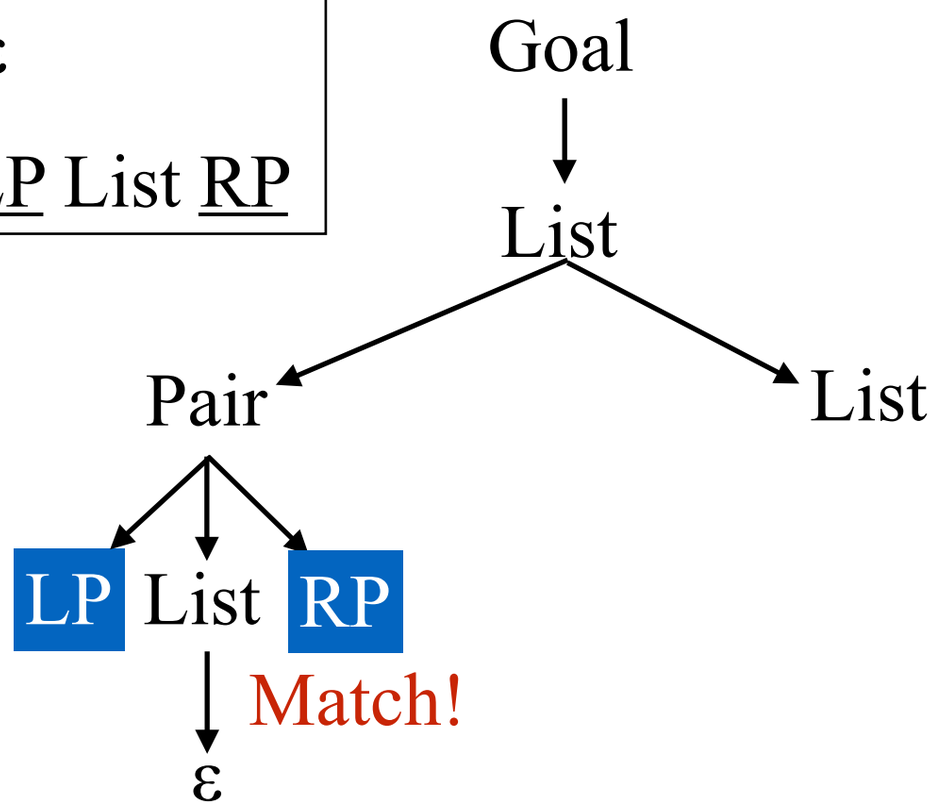
Sentential Form:
LP RP List

Applied Production:
3. List ::= ε

LL(1) Parsing Example

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

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



Remaining Input:
RP LP LP RP RP

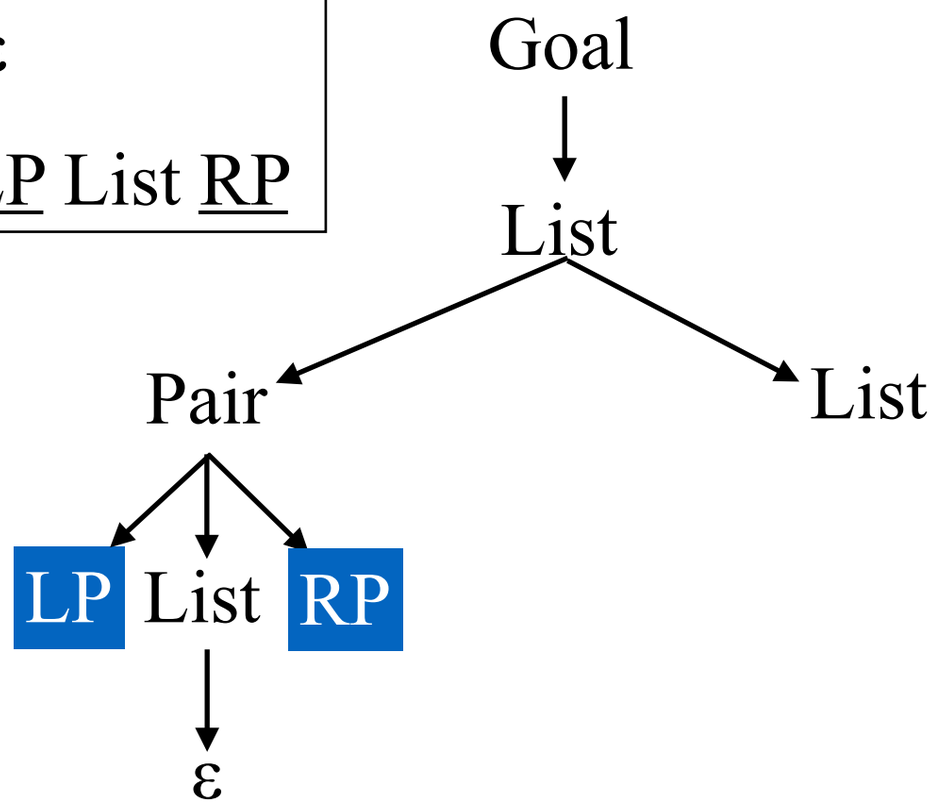
Sentential Form:
LP RP List

Applied Production:

LL(1) Parsing Example

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

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



Remaining Input:
LP LP RP RP

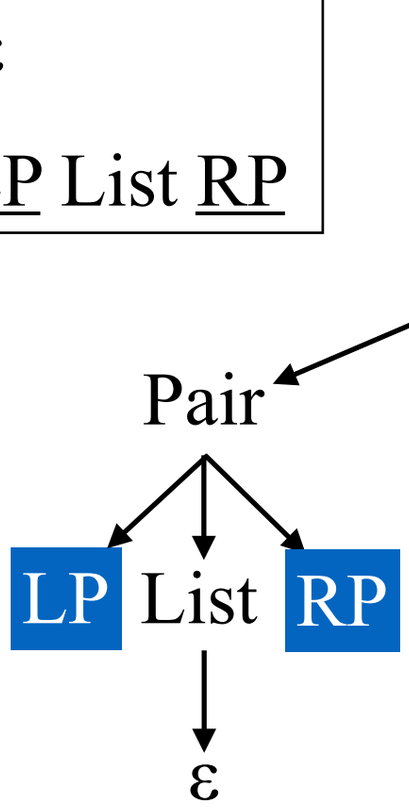
Sentential Form:
LP RP List

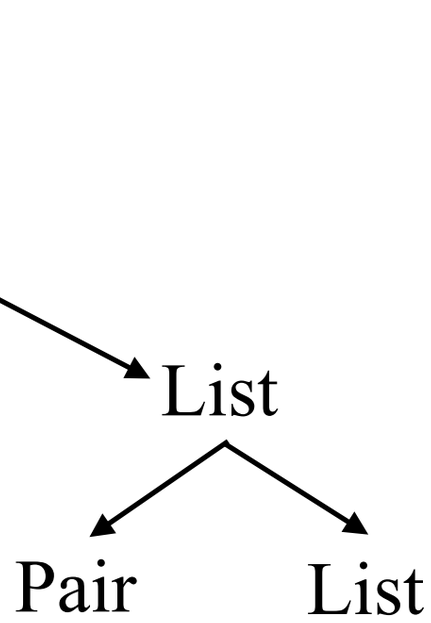
Applied Production:

LL(1) Parsing Example

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

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





Remaining Input:
LP LP RP RP

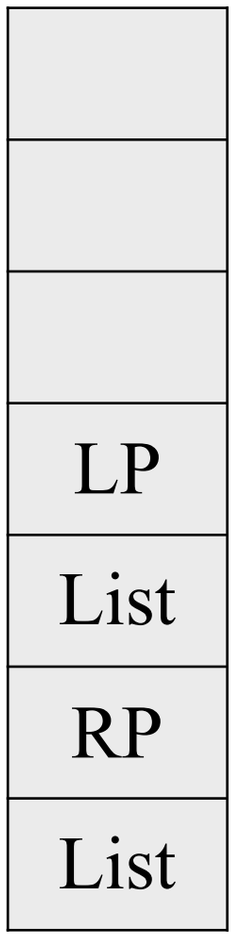
Sentential Form:
LP RP Pair List

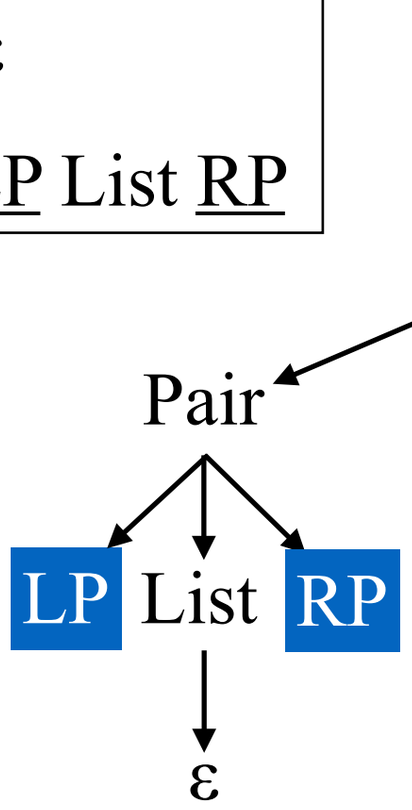
Applied Production:
2. List ::= Pair List

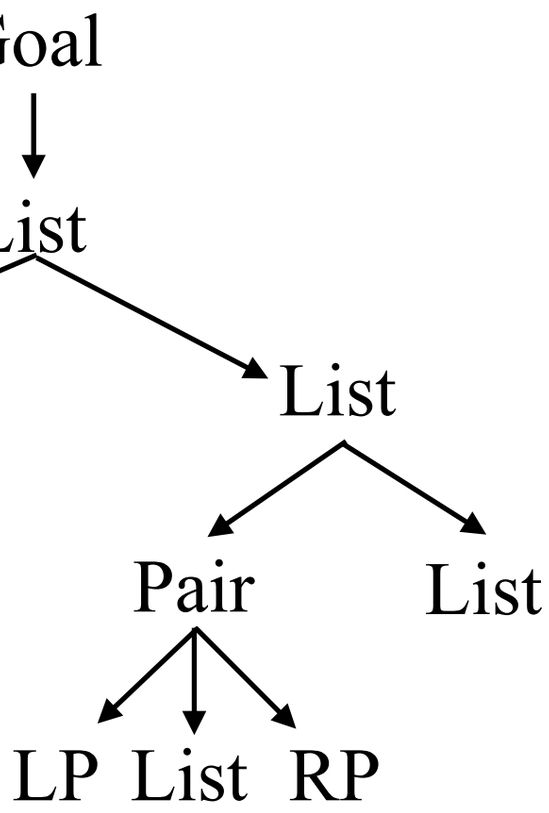
LL(1) Parsing Example

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

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







Remaining Input:
LP LP RP RP

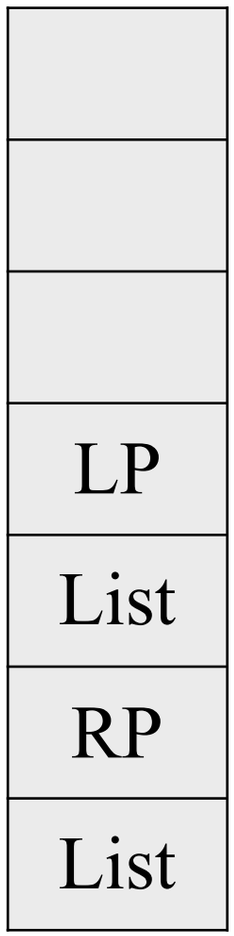
Sentential Form:
LP RP LP List RP List

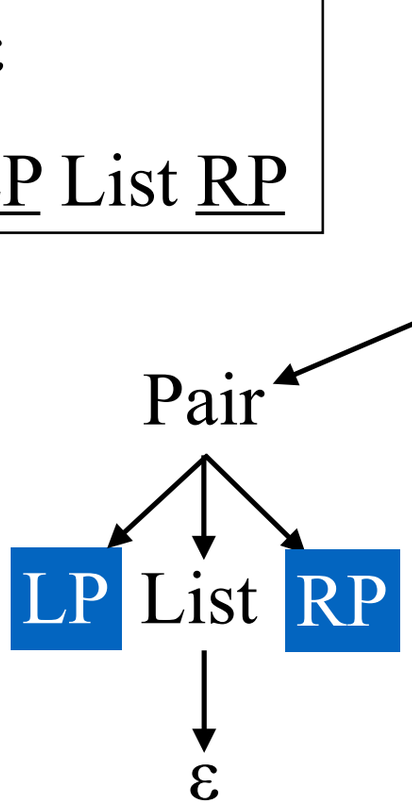
Applied Production:
4. Pair ::= LP List RP

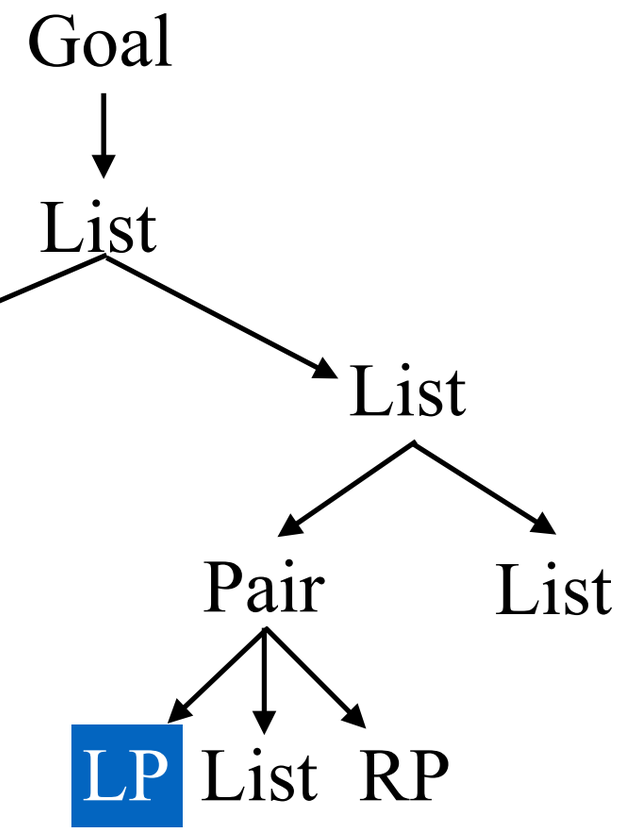
LL(1) Parsing Example

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

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







Match!

Remaining Input:
LP LP RP RP

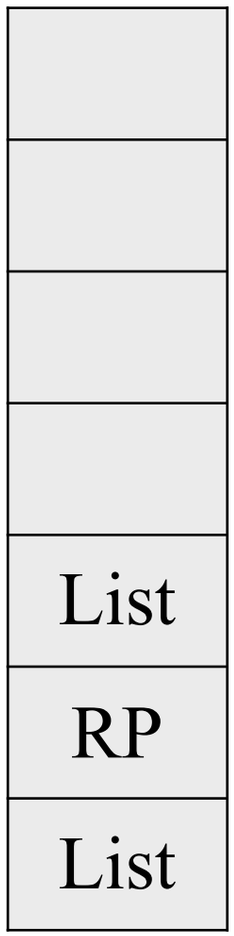
Sentential Form:
LP RP LP List RP List

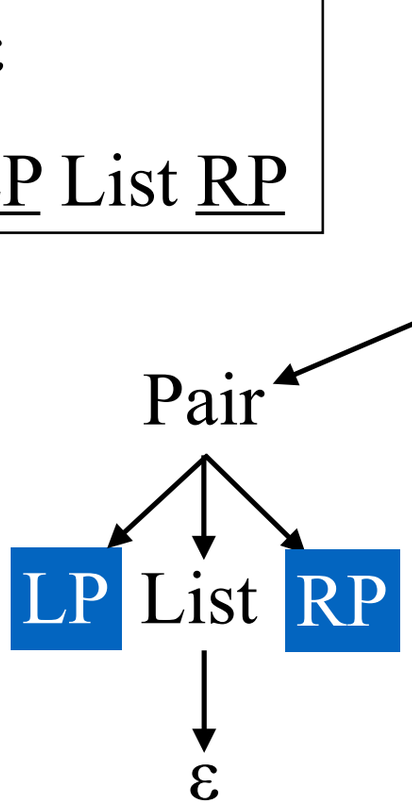
Applied Production:

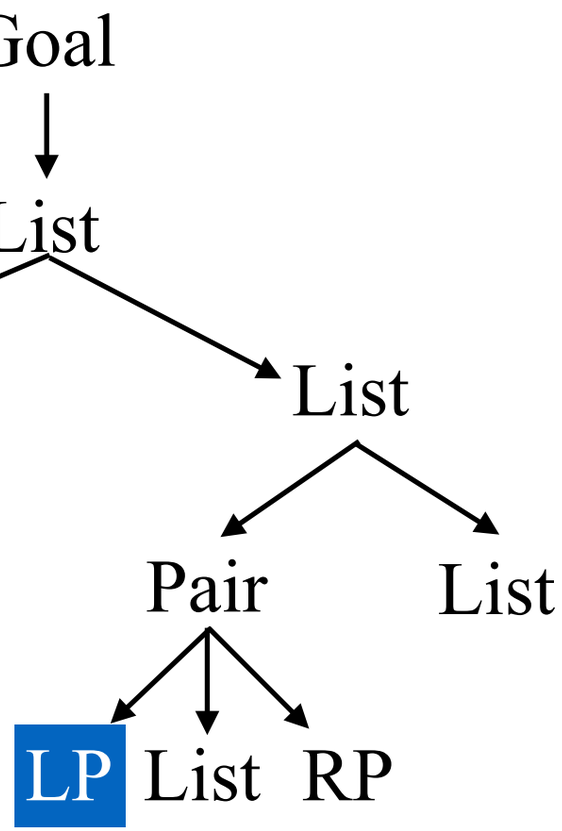
LL(1) Parsing Example

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

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







Remaining Input:
LP RP RP

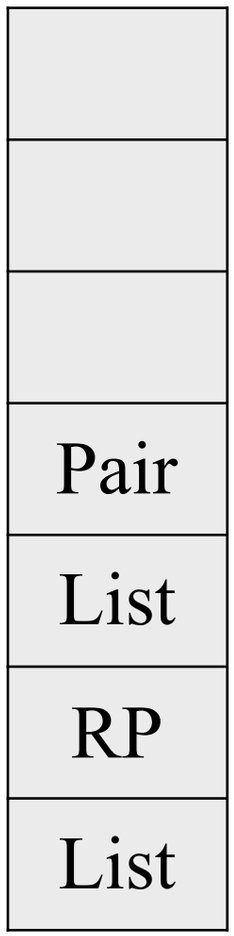
Sentential Form:
LP RP LP List RP List

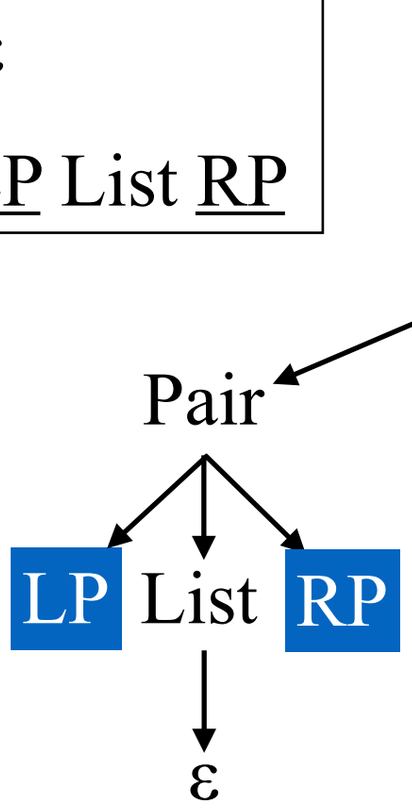
Applied Production:

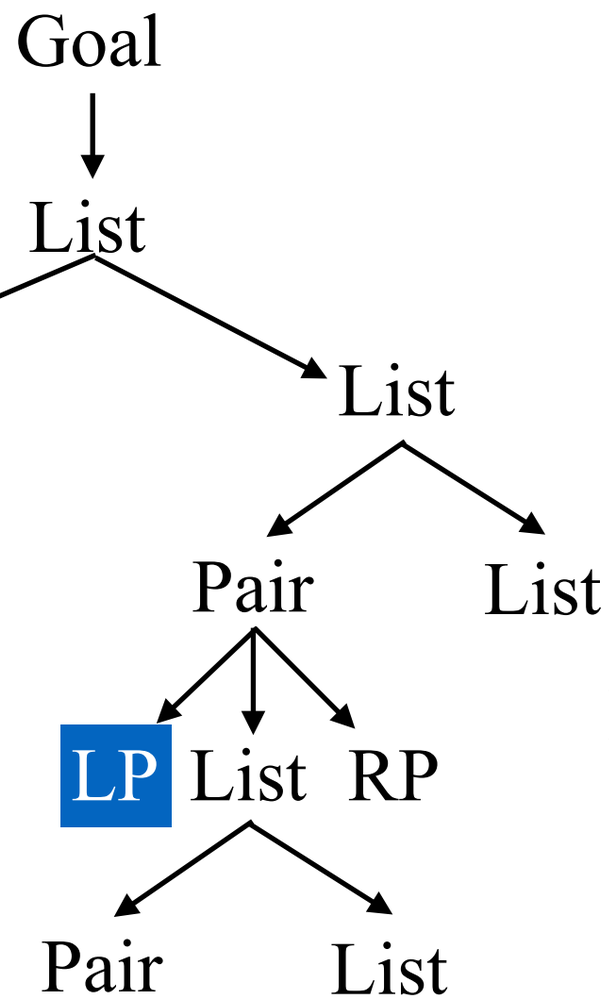
LL(1) Parsing Example

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

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







Remaining Input:
LP RP RP

Sentential Form:
LP RP LP Pair List RP List

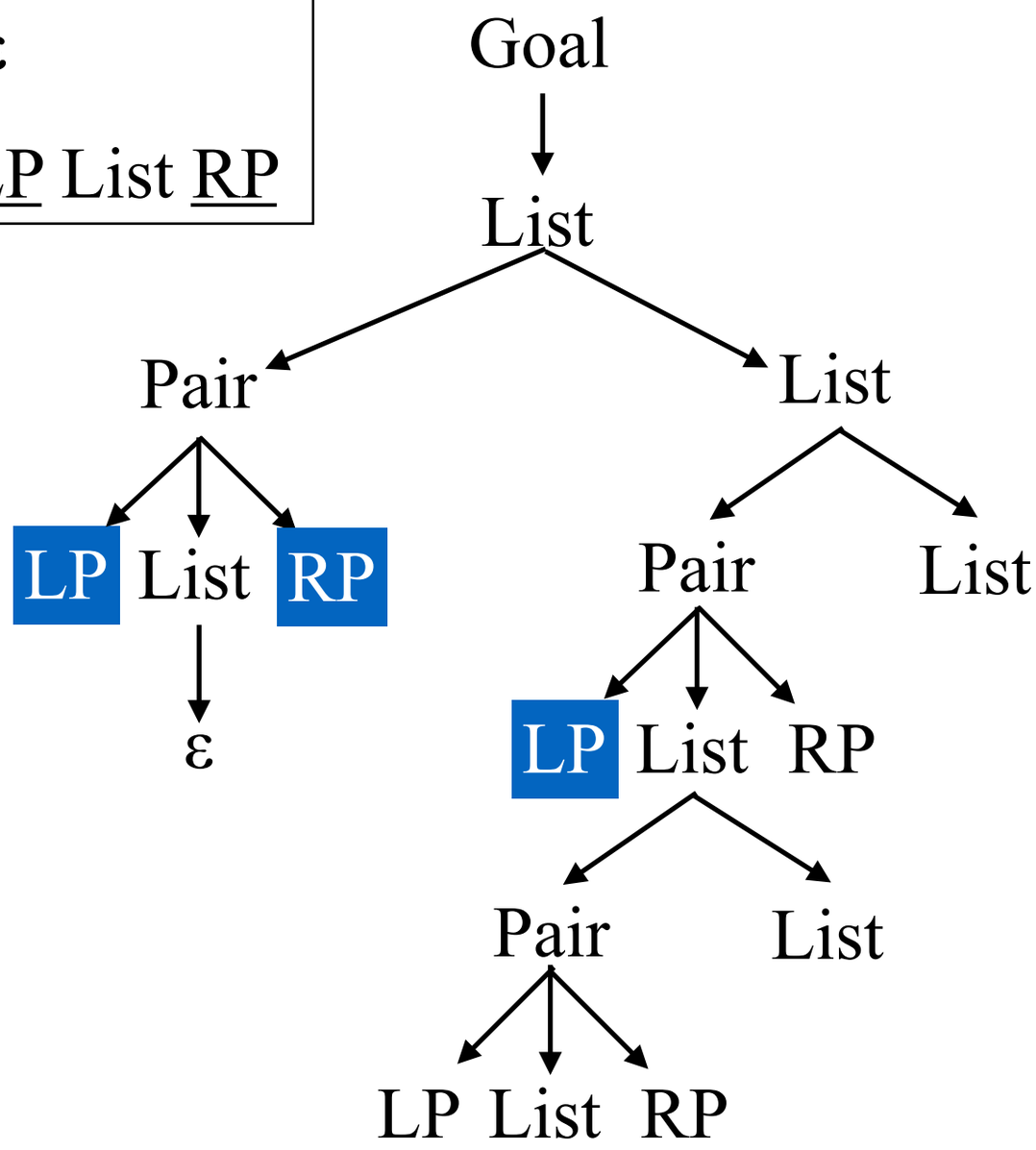
Applied Production:
2. List ::= Pair List

LL(1) Parsing Example

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

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

LP
List
RP
List
RP
List



Remaining Input:
LP RP RP

Sentential Form:
LP RP LP
LP List RP List RP List

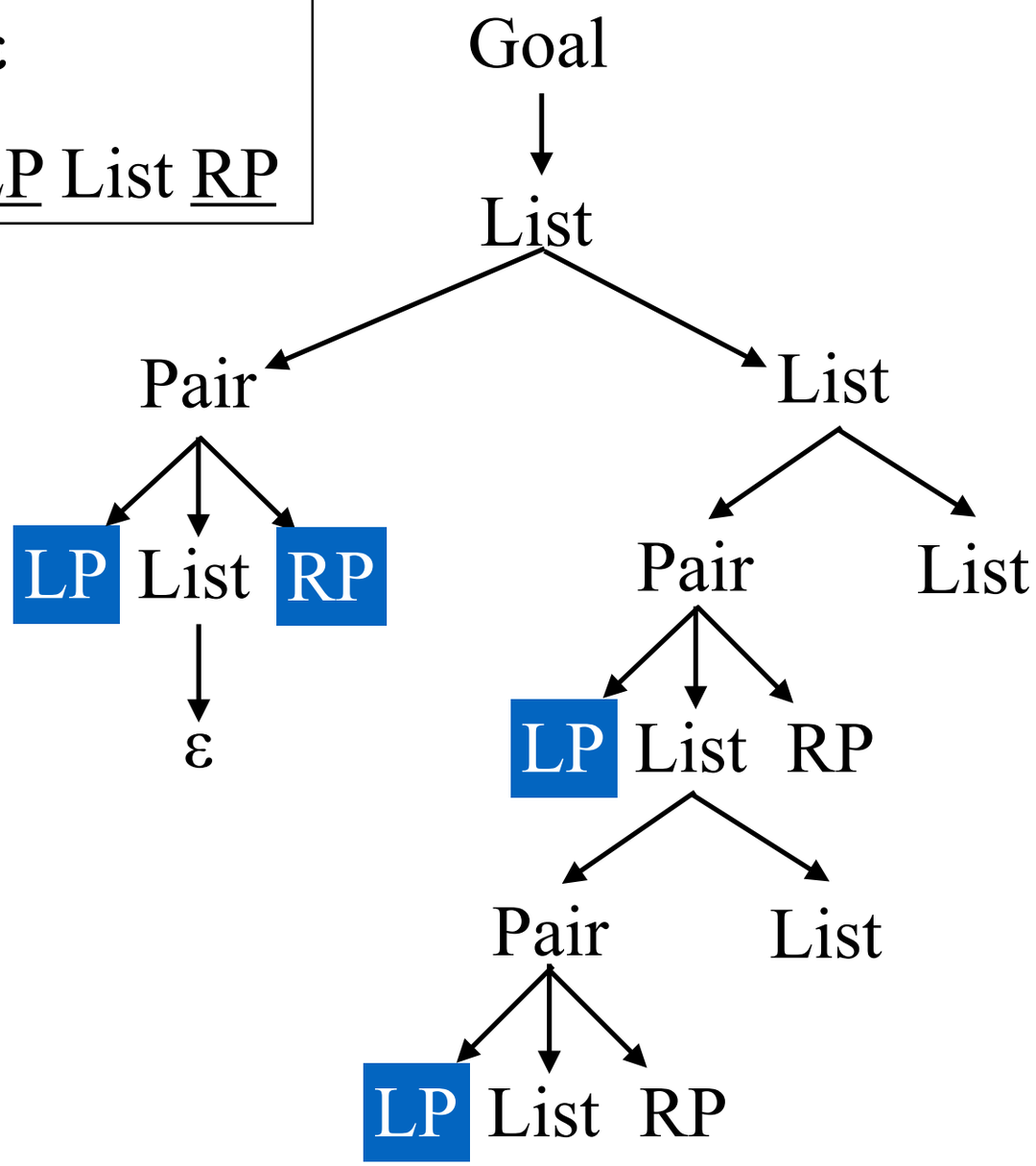
Applied Production:
4. Pair ::= LP List RP

LL(1) Parsing Example

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

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

LP
List
RP
List
RP
List



Match!

Remaining Input:
LP RP RP

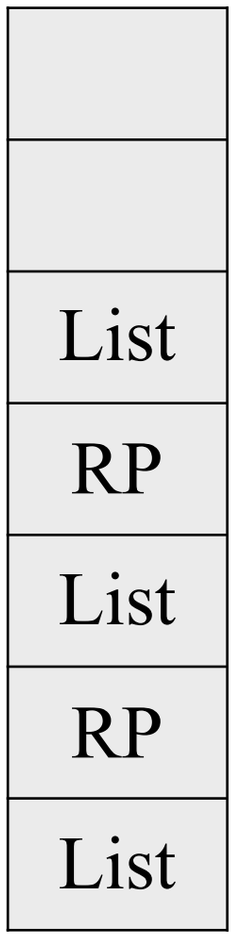
Sentential Form:
LP RP LP
LP List RP List RP List

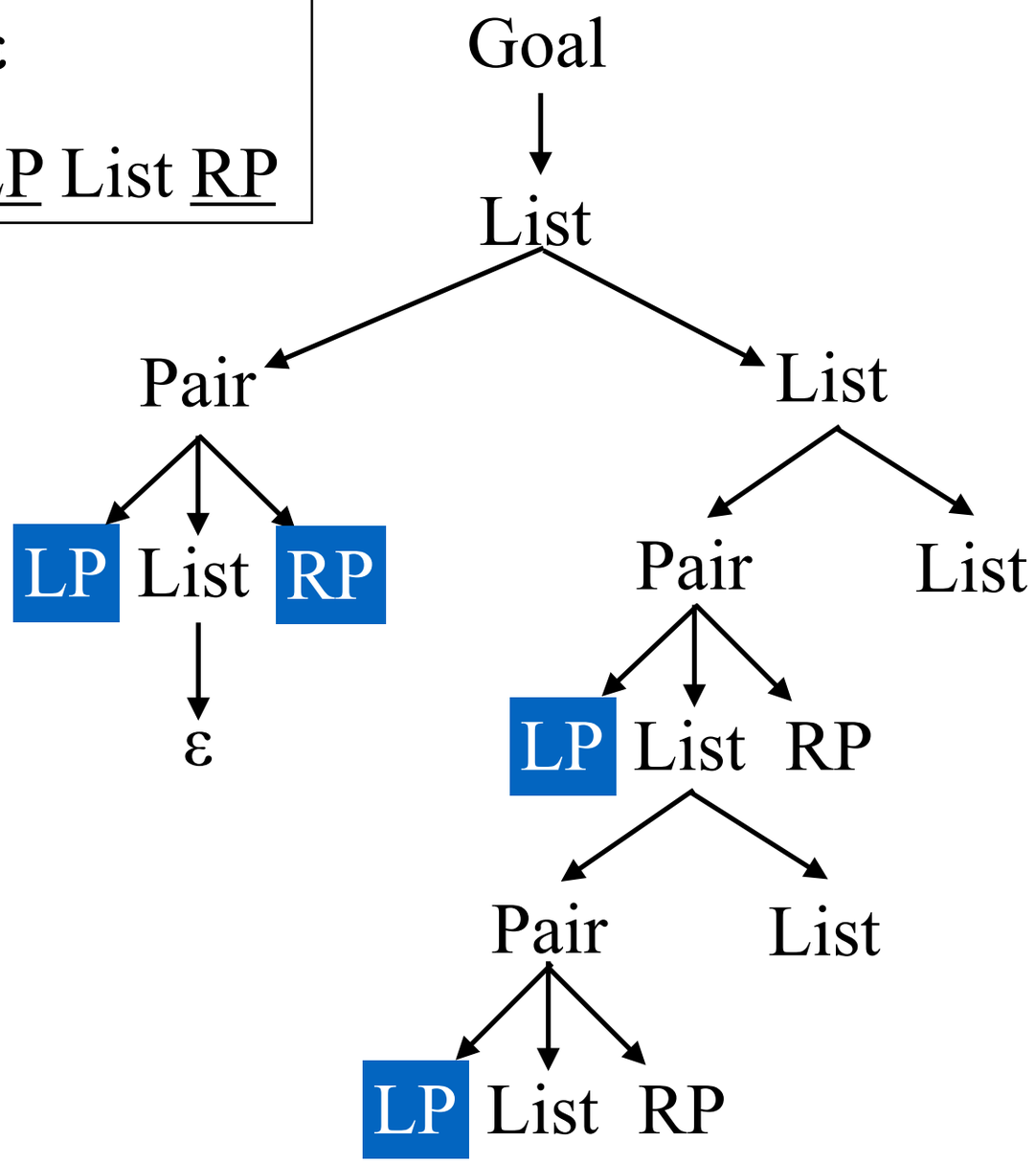
Applied Production:

LL(1) Parsing Example

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

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





Remaining Input:
RP RP

Sentential Form:
LP RP LP
LP List RP List RP List

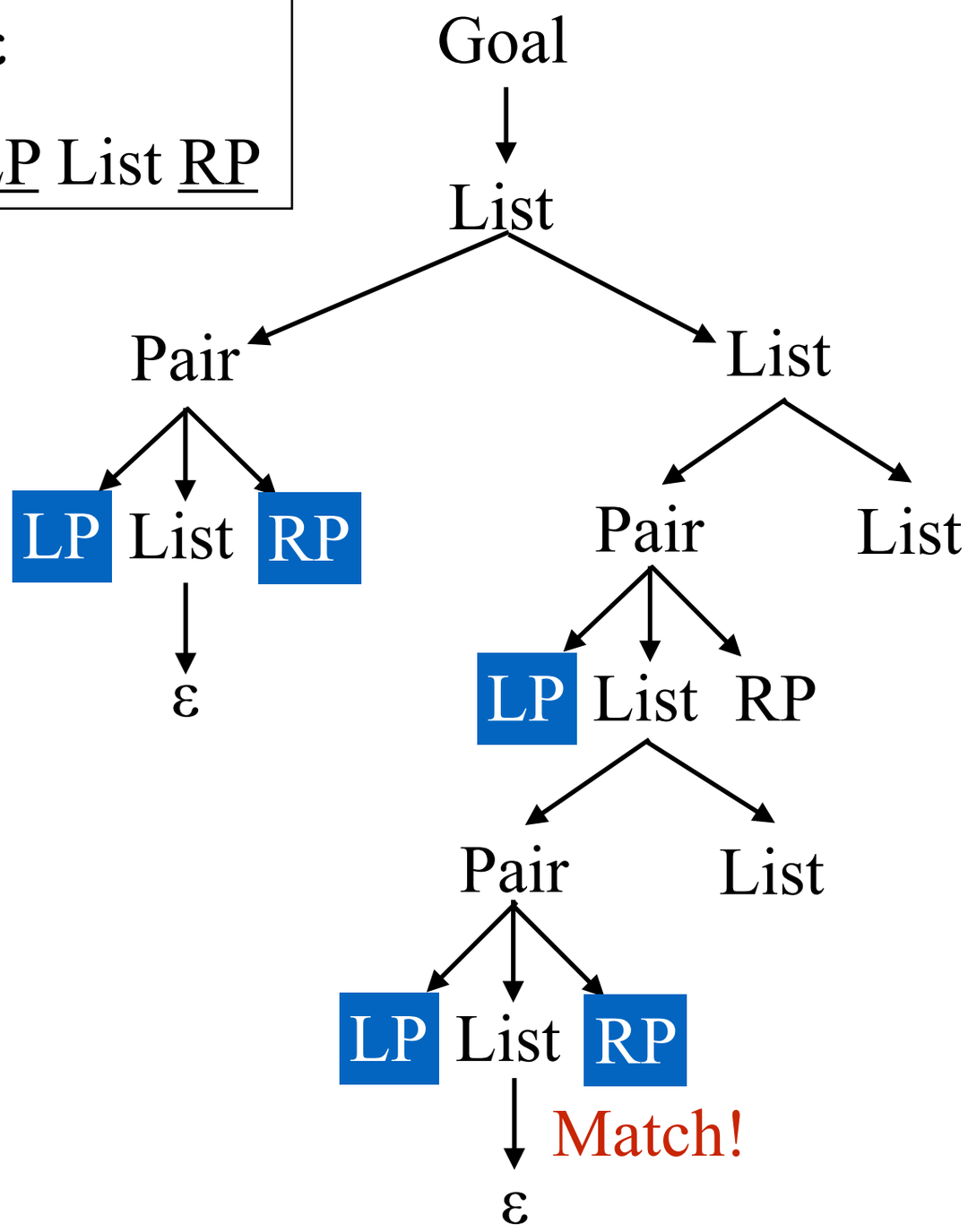
Applied Production:

LL(1) Parsing Example

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

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

RP
List
RP
List



Remaining Input:
RP RP

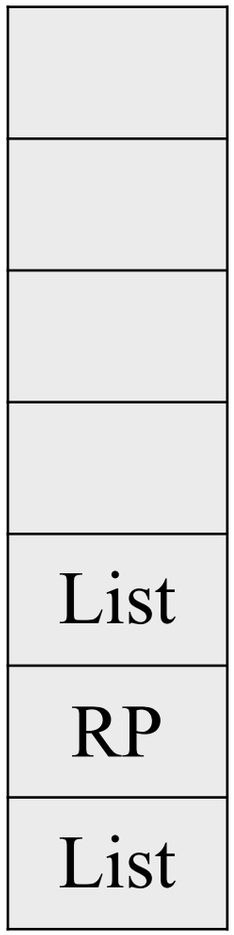
Sentential Form:
LP RP LP
LP RP List RP List

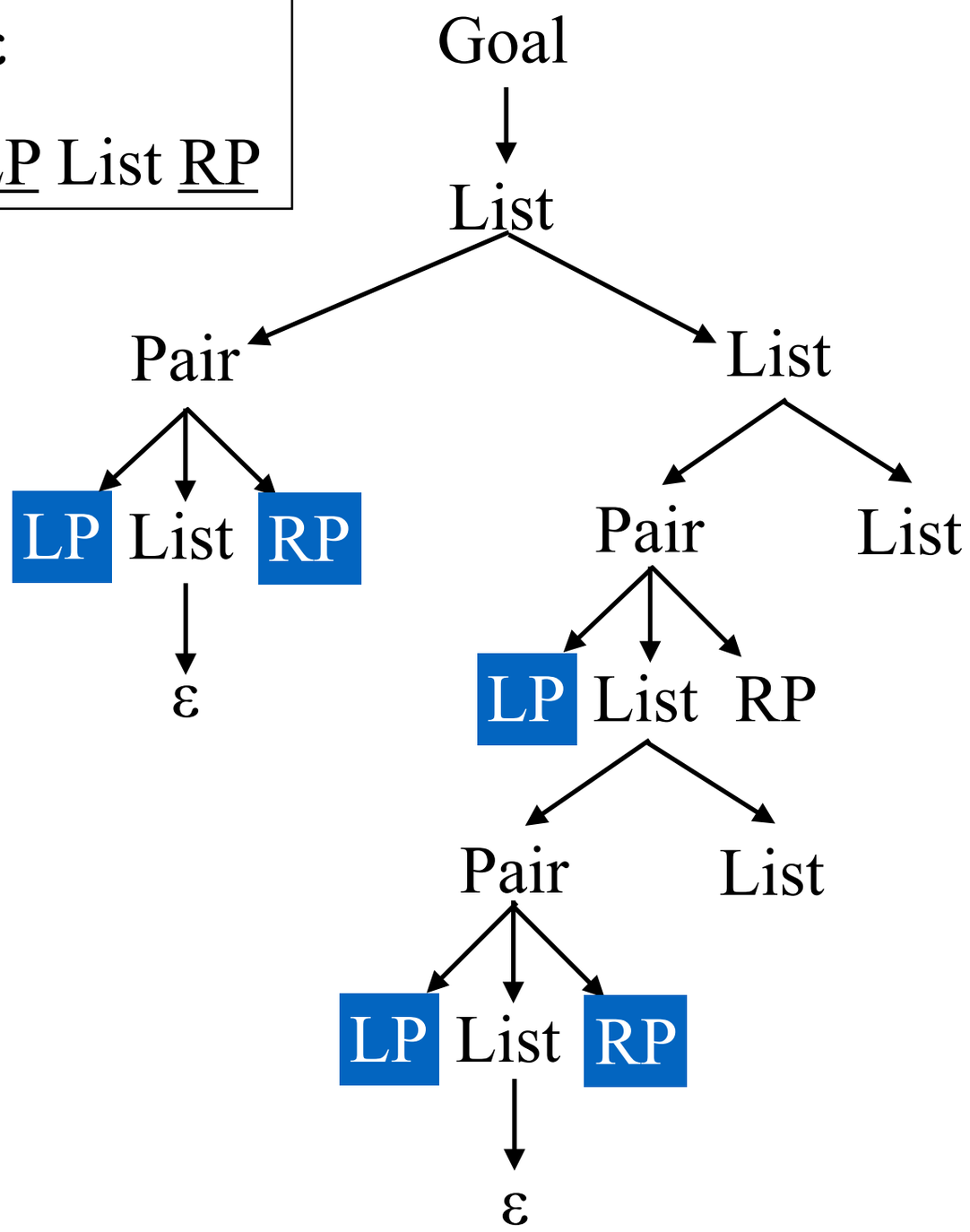
Applied Production:

LL(1) Parsing Example

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

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





Remaining Input:
RP

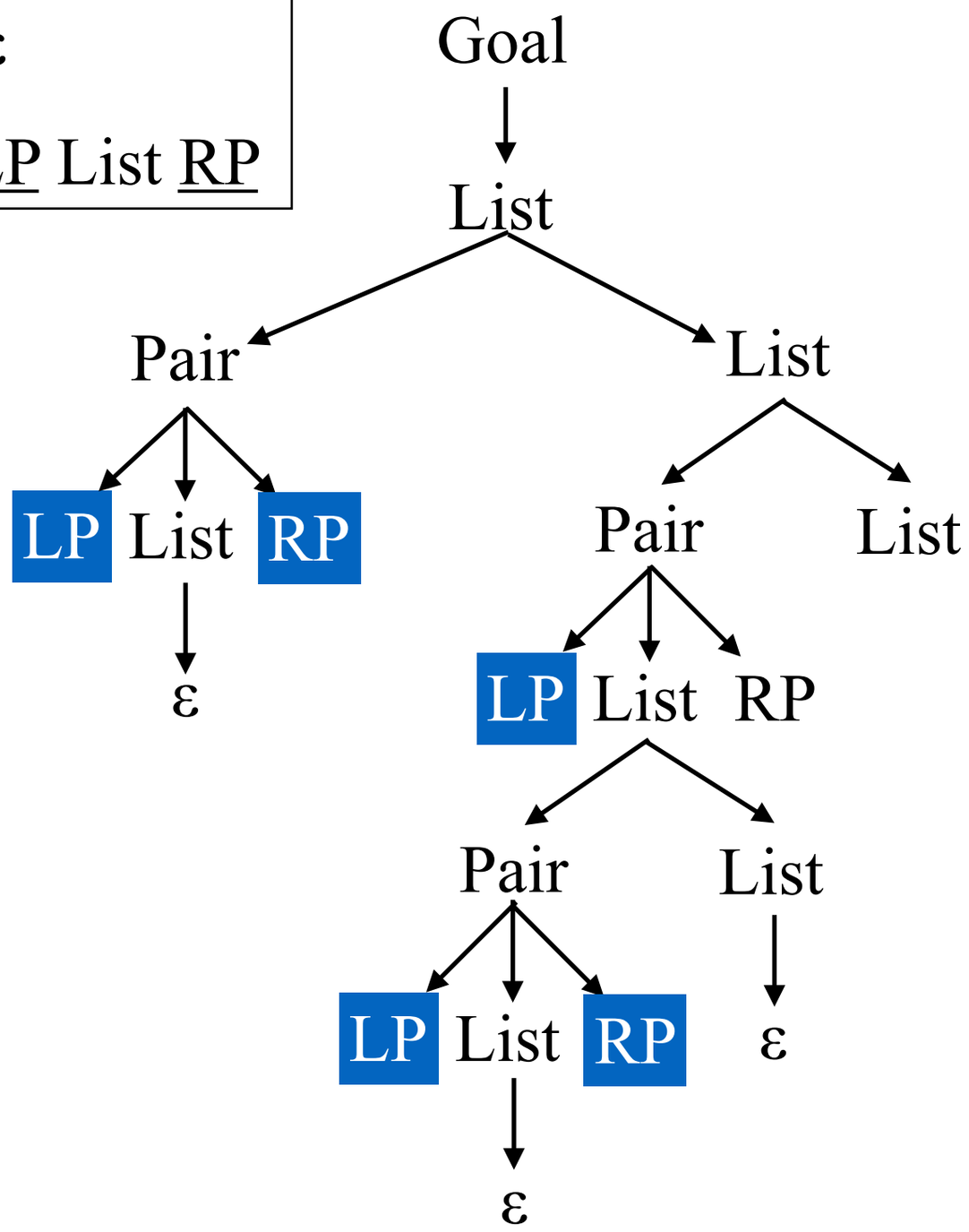
Sentential Form:
LP RP LP
LP RP List RP List

Applied Production:

LL(1) Parsing Example

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

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



Remaining Input:
RP

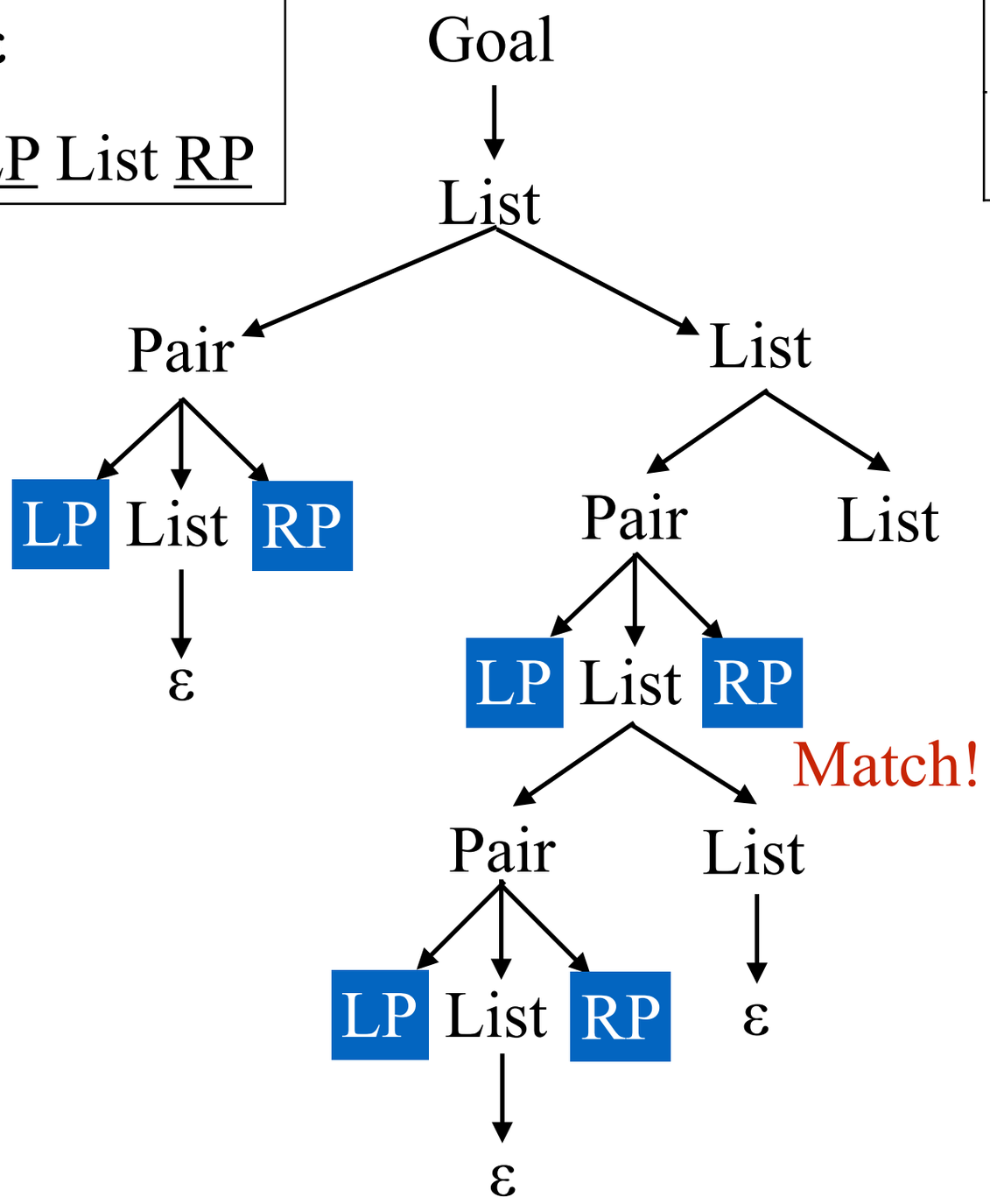
Sentential Form:
LP RP LP
LP RP RP List

Applied Production:
3. List ::= ε

LL(1) Parsing Example

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

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



Remaining Input:
RP

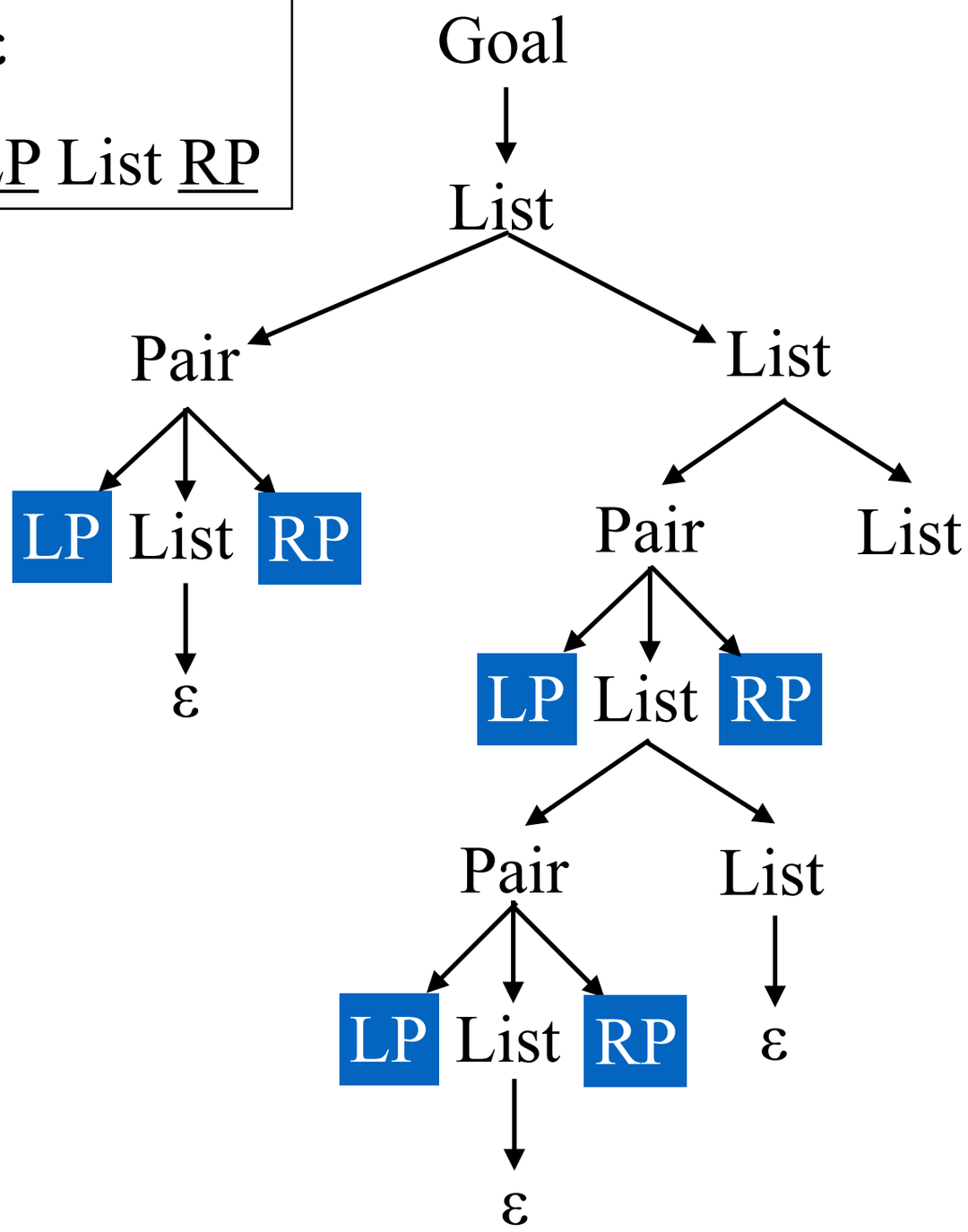
Sentential Form:
LP RP LP
LP RP RP List

Applied Production:

LL(1) Parsing Example

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

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



Remaining Input:

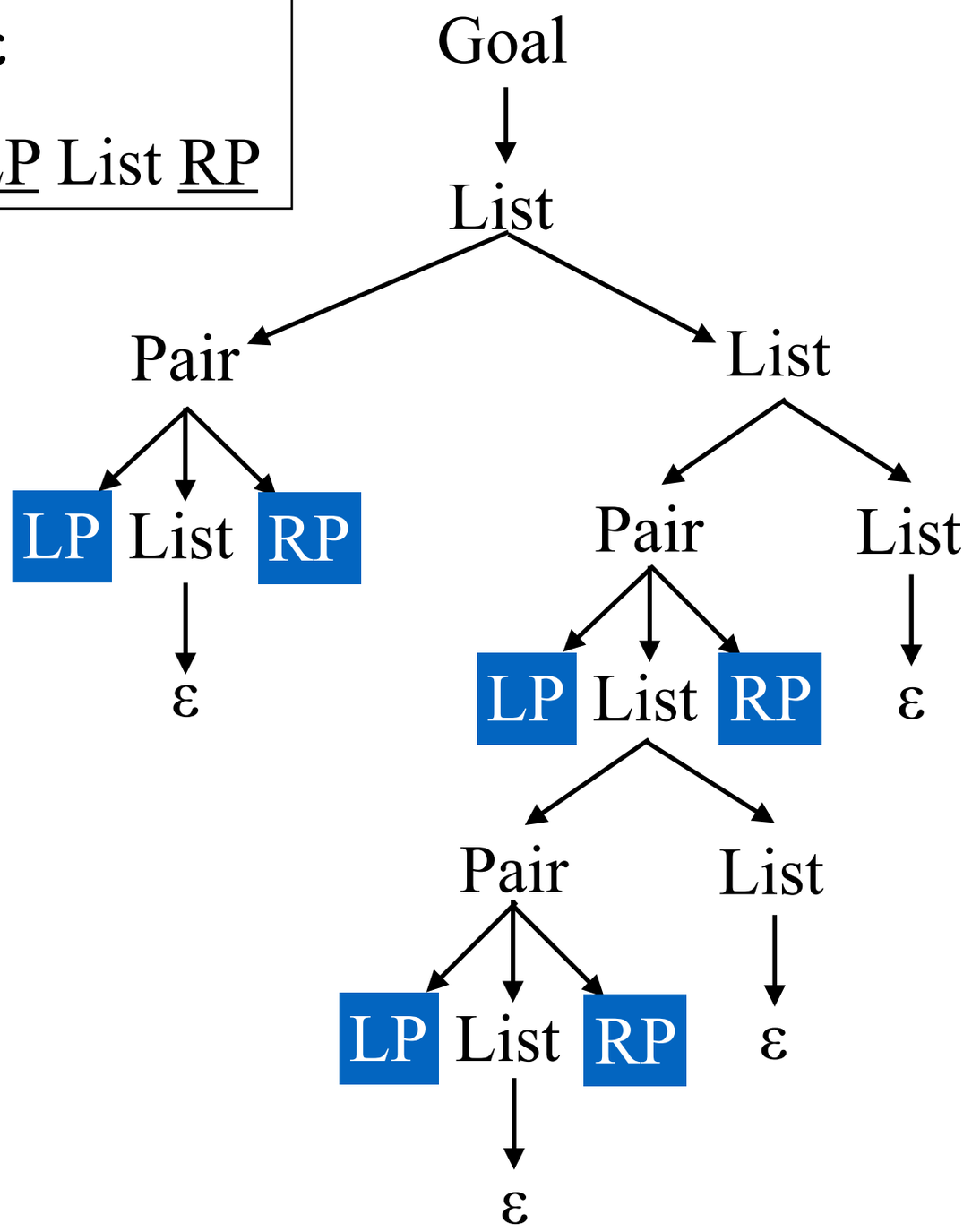
Sentential Form:
LP RP LP
LP RP RP List

Applied Production:

LL(1) Parsing Example

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

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



Remaining Input:

Sentential Form:
LP RP LP LP RP RP

Applied Production:
3. List ::= ε

Recursive Descent Parsing

Recursive descent parser for LL(1)

- Each **non-terminal** has an associated parsing procedure that can recognize any sequence of tokens generated by that **non-terminal**
- There is a main routine to initialize all globals (e.g:the *token variable* in previous code example) and call the start symbol. On return, check whether token==EOF, and whether errors occurred.
- Within a parsing procedure, both **non-terminals** and **terminals** can be matched:
  - ➡ Non-terminal A: call procedure for A
  - ➡ Token t: compare t with current input token;
    if matched, **consume input**, otherwise, ERROR
- Parsing procedure may contain code that performs some useful "computations" (*syntax directed translation)*

Recursive Descent Parsing (pseudo code)

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

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

```
main:  {
       token := next_token( );
       if ( List( ) and token == EOF) print “accept” else print “error”;
}
```

Recursive Descent Parsing (pseudo code)

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

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

```
bool List( ):  {
  switch token {
     case LP:
                  call Pair( );
                  call List( );
                 break;
     case RP:
     case EOF:  return true;
                  break;
     default: return false;
  }
  return true;
}
```

```
bool Pair( ):  {
  switch token {
    case LP:  token := next_token( );
                  call List( );
                  if ( token == RP ) {
                     token := next_token( );
                     return true;
                  }
                  else
                     return false;
                  break;
    default: return false;
  }
}
```

Syntax Directed Translation

Examples:

- Interpreter
- Code generator
- Type checker
- Performance estimator

Use hand-written recursive descent LL(1) parser

Example: the Original Parser

1: < expr > ::= + < expr > < expr > |
2: < digit >
3: < digit > ::= 0 | 1 | 2 | 3 | … | 9

	+	0…9	other
< expr >	rule 1	rule 2	error
< digit >	error	rule 3	error

```
bool expr( ) {

        switch token {
                case +:       token := next_token( );
                              expr( ); expr( ); break;
                case 0..9:    digit( ); break;
                …
          }
        }
bool digit( ) {  // return value of constant
        switch token {
                case 1: token := next_token( ); break;
                case 2: token := next_token( ); break;
              …
        }
        }
```

Example: the Original Parser

1: < expr > ::= + < expr > < expr > |
2: < digit >
3: < digit > ::= 0 | 1 | 2 | 3 | … | 9

	+	0…9	other
< expr >	rule 1	rule 2	error
< digit >	error	rule 3	error

call <expr>

What happens when you parse expression
“ + **2** + **1 2**”

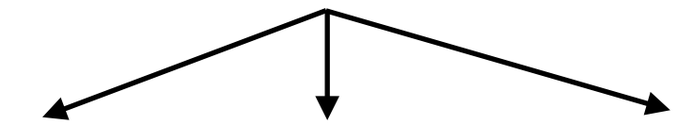
```
bool expr( ):  // return value of the expression

        switch token {
                case +:       token := next_token( );
                               expr( );
                               expr( ); break;
                case 0..9:    digit( ); break;
                …
        }

bool digit( ):  // return value of constant
        switch token {
                case 1: token := next_token( ); break;
                case 2: token := next_token( ); break;
              …
        }
```

Example: the Original Parser

1: < expr > ::= + < expr > < expr > |
2: < digit >
3: < digit > ::= 0 | 1 | 2 | 3 | … | 9

	+	0…9	other
< expr >	rule 1	rule 2	error
< digit >	error	rule 3	error

call <expr>



\+ call <expr>

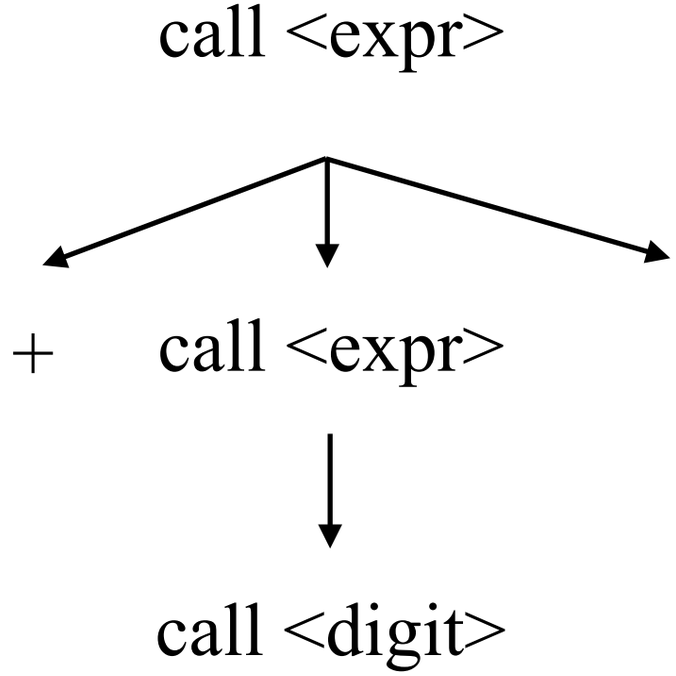
What happens when you parse expression
“ + **2** + **1 2**”

```
bool expr( ):  // return value of the expression

        switch token {
                case +:       token := next_token( );
                               expr( );
                               expr( ); break;
                case 0..9:    digit( ); break;
                …
        }

bool digit( ):  // return value of constant
        switch token {
                case 1: token := next_token( ); break;
                case 2: token := next_token( ); break;
              …
        }
```

Example: the Original Parser

1: < expr > ::= + < expr > < expr > |
2: < digit >
3: < digit > ::= 0 | 1 | 2 | 3 | … | 9

	+	0…9	other
< expr >	rule 1	rule 2	error
< digit >	error	rule 3	error

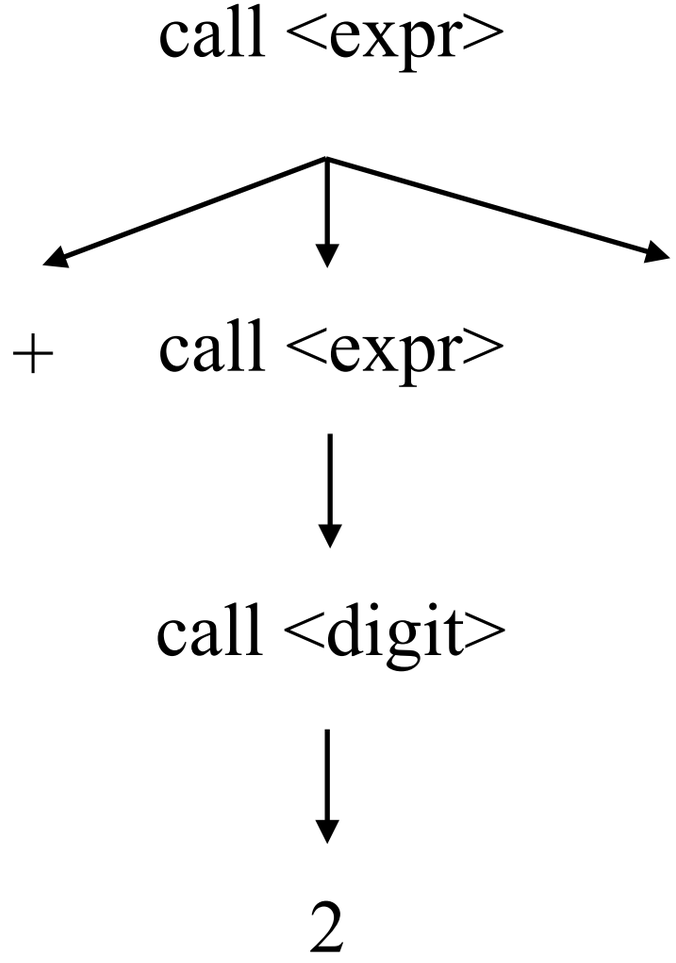


What happens when you parse expression
“ + **2** + **1** **2**”

Example: the Original Parser

1: < expr > ::= + < expr > < expr > |
2: < digit >
3: < digit > ::= 0 | 1 | 2 | 3 | … | 9

	+	0…9	other
< expr >	rule 1	rule 2	error
< digit >	error	rule 3	error

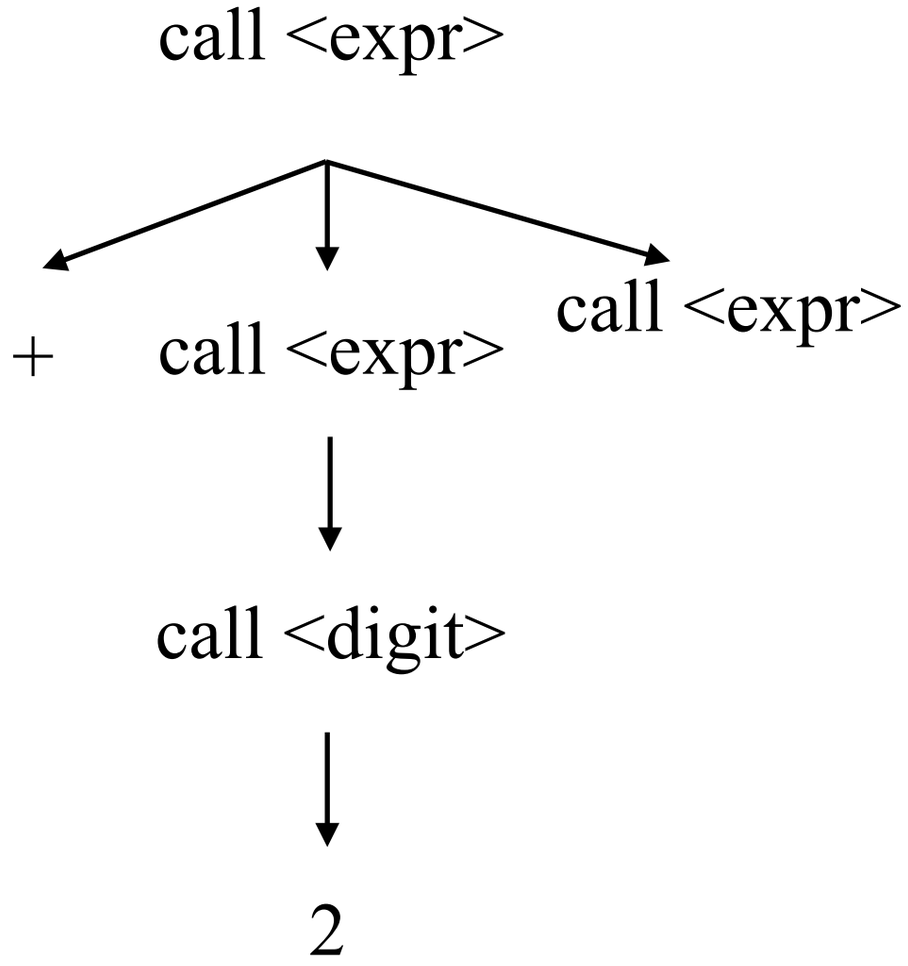


What happens when you parse expression
“ + **2** + **1** **2**”

Example: the Original Parser

1: < expr > ::= + < expr > < expr > |
2: < digit >
3: < digit > ::= 0 | 1 | 2 | 3 | … | 9

	+	0…9	other
< expr >	rule 1	rule 2	error
< digit >	error	rule 3	error

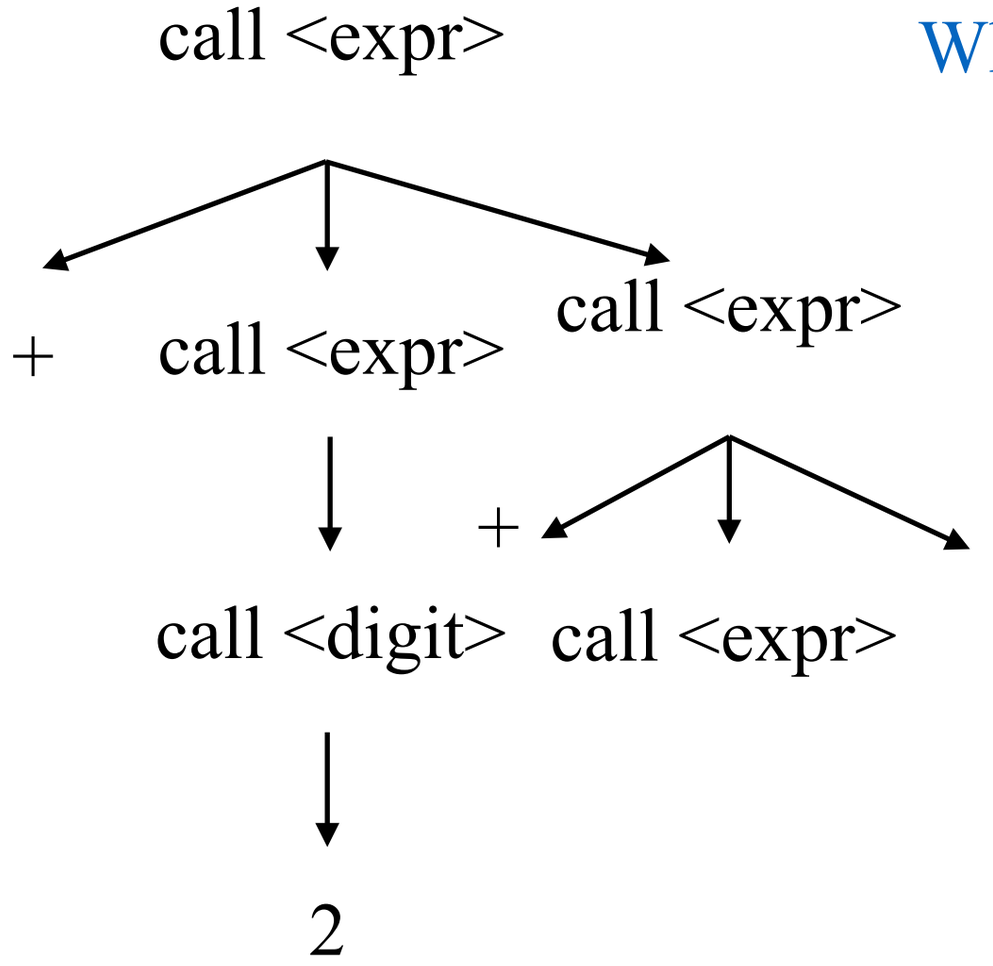


What happens when you parse expression
“ + **2** + **1 2**”

Example: the Original Parser

1: < expr > ::= + < expr > < expr > |
2: < digit >
3: < digit > ::= 0 | 1 | 2 | 3 | … | 9

	+	0…9	other
< expr >	rule 1	rule 2	error
< digit >	error	rule 3	error

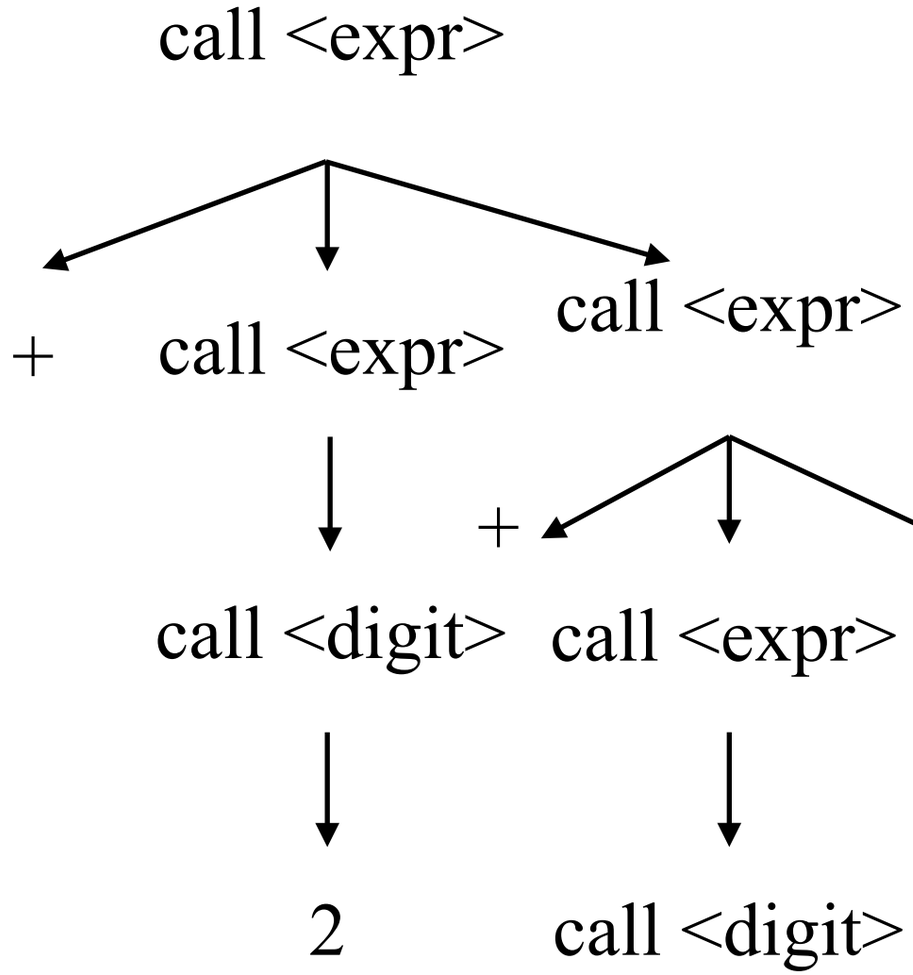


What happens when you parse expression
“ + **2** + **1 2**”

Example: the Original Parser

1: < expr > ::= + < expr > < expr > |
2: < digit >
3: < digit > ::= 0 | 1 | 2 | 3 | … | 9

	+	0…9	other
< expr >	rule 1	rule 2	error
< digit >	error	rule 3	error

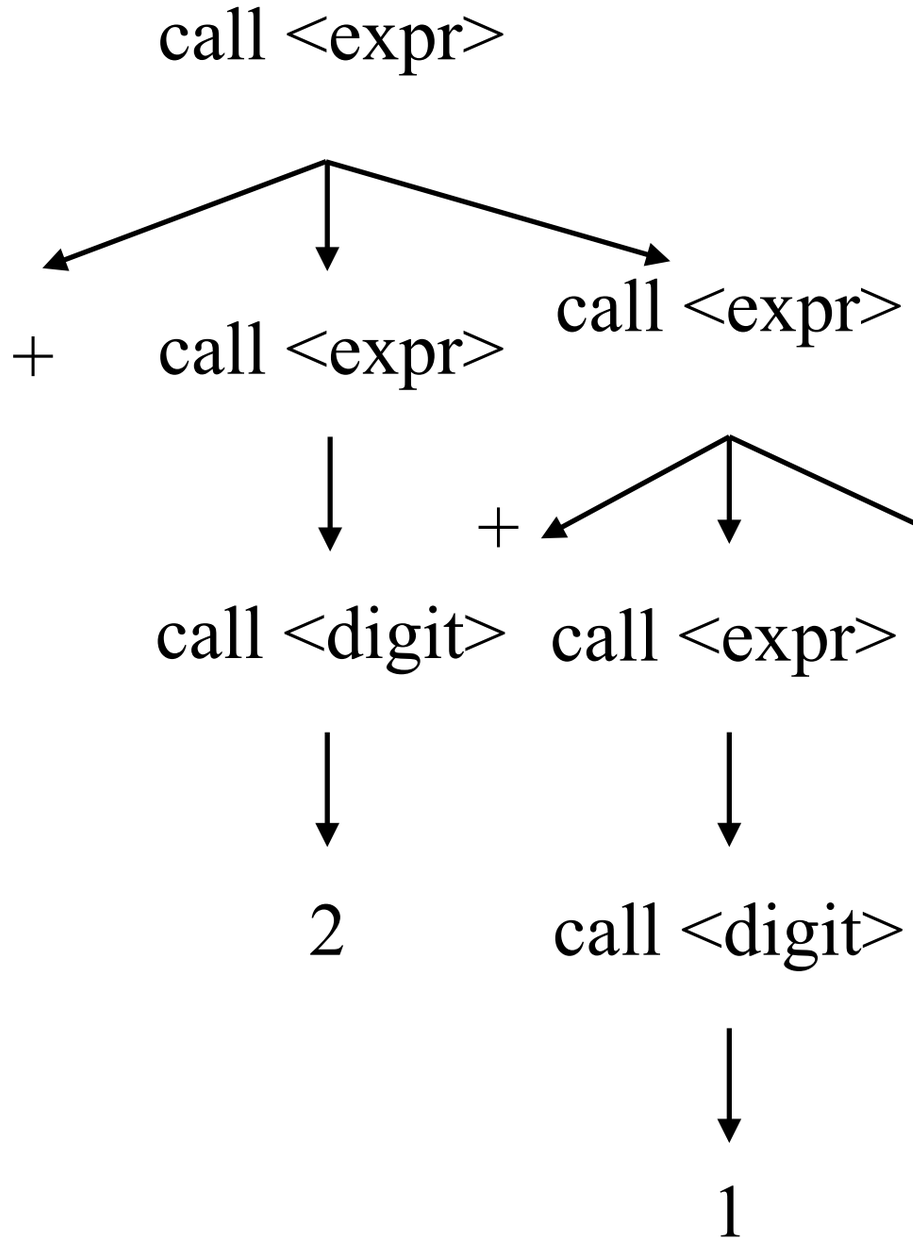


What happens when you parse expression
“ + **2** + **1 2**”

Example: the Original Parser

1: < expr > ::= + < expr > < expr > |
2: < digit >
3: < digit > ::= 0 | 1 | 2 | 3 | … | 9

	+	0…9	other
< expr >	rule 1	rule 2	error
< digit >	error	rule 3	error

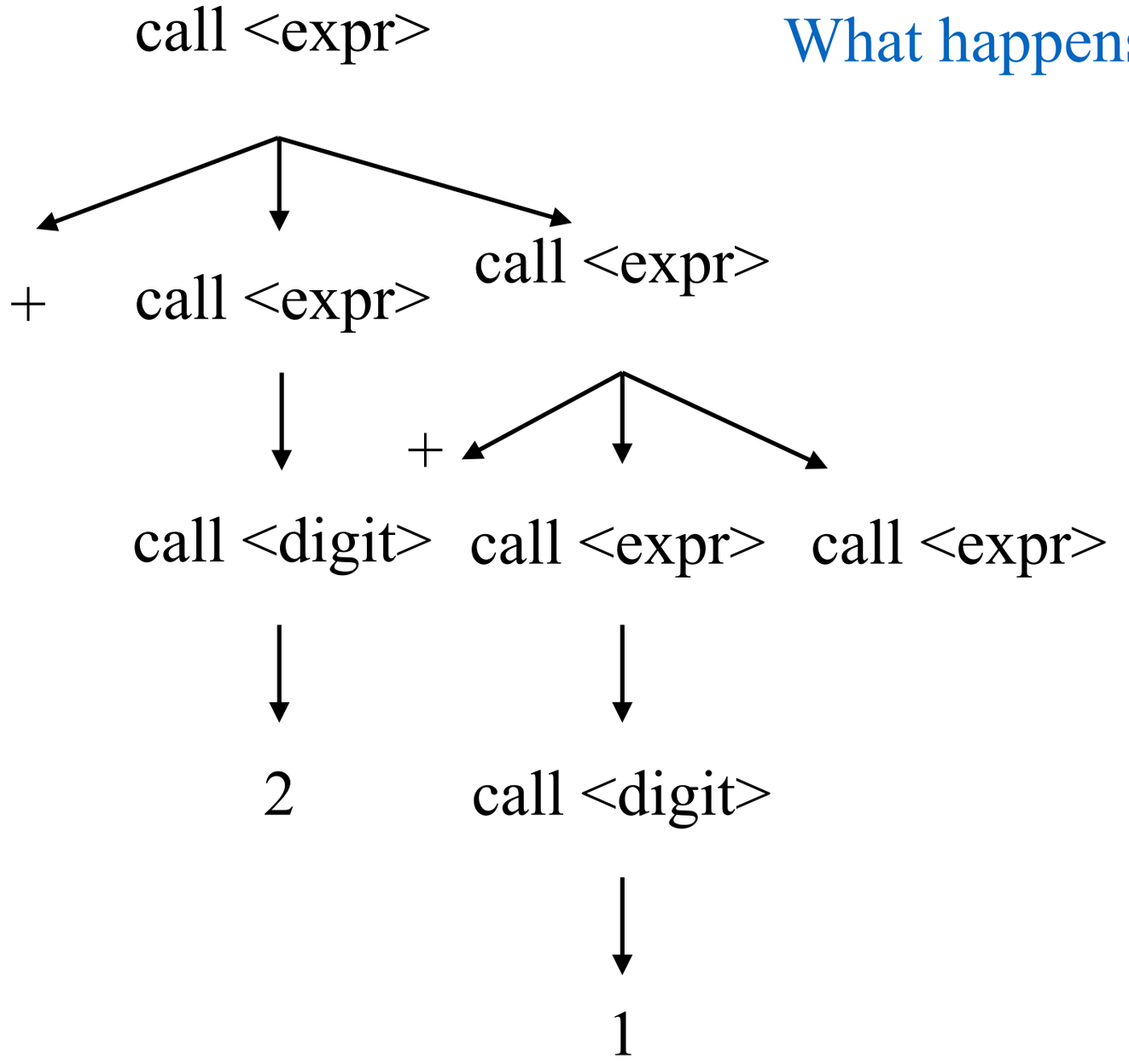


What happens when you parse expression
“ + **2** + **1** **2**”

Example: the Original Parser

1: < expr > ::= + < expr > < expr > |
2: < digit >
3: < digit > ::= 0 | 1 | 2 | 3 | … | 9

	+	0…9	other
< expr >	rule 1	rule 2	error
< digit >	error	rule 3	error

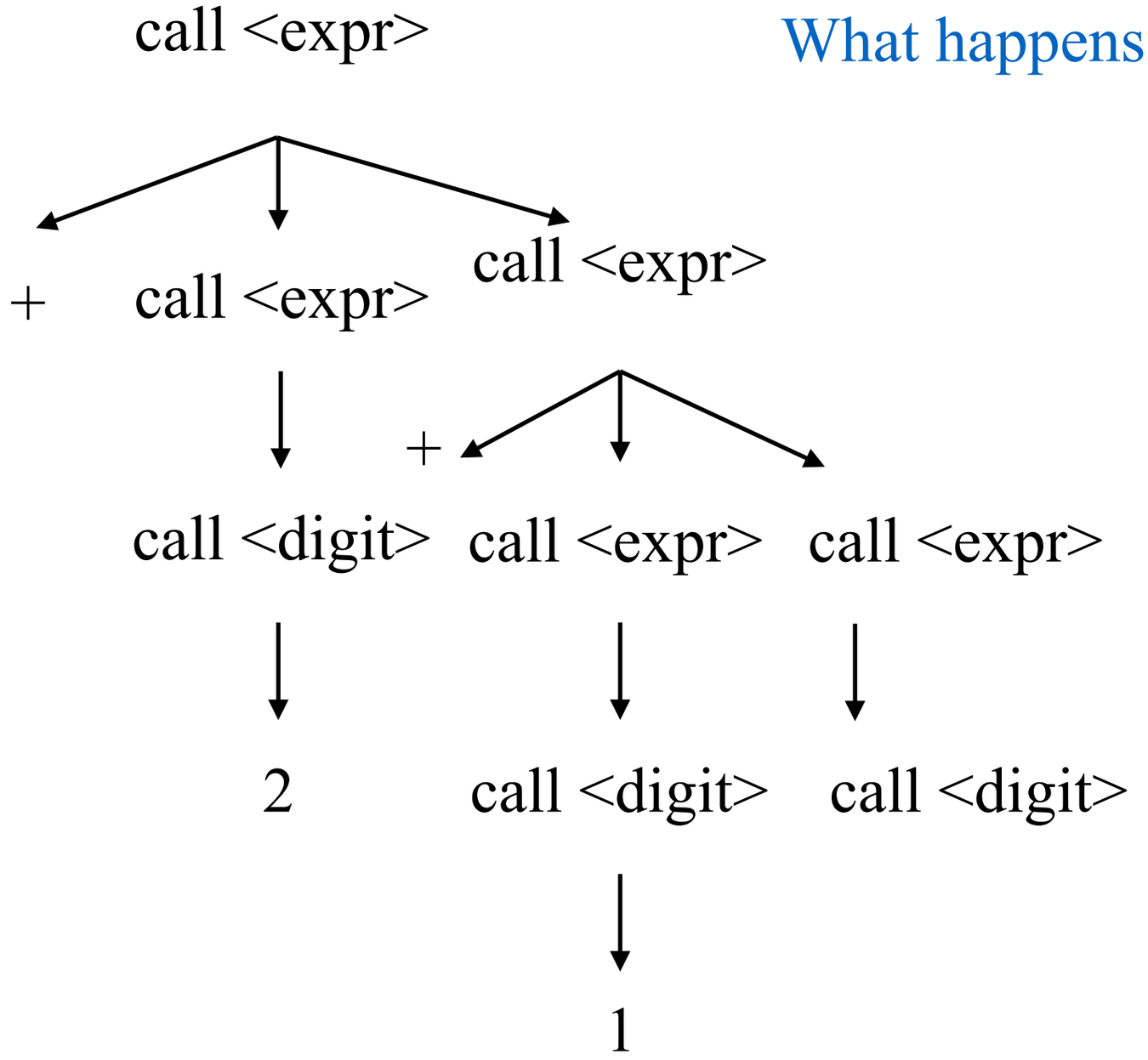


What happens when you parse expression
“ + **2** + **1** **2**”

Example: the Original Parser

1: < expr > ::= + < expr > < expr > |
2: < digit >
3: < digit > ::= 0 | 1 | 2 | 3 | … | 9

	+	0…9	other
< expr >	rule 1	rule 2	error
< digit >	error	rule 3	error

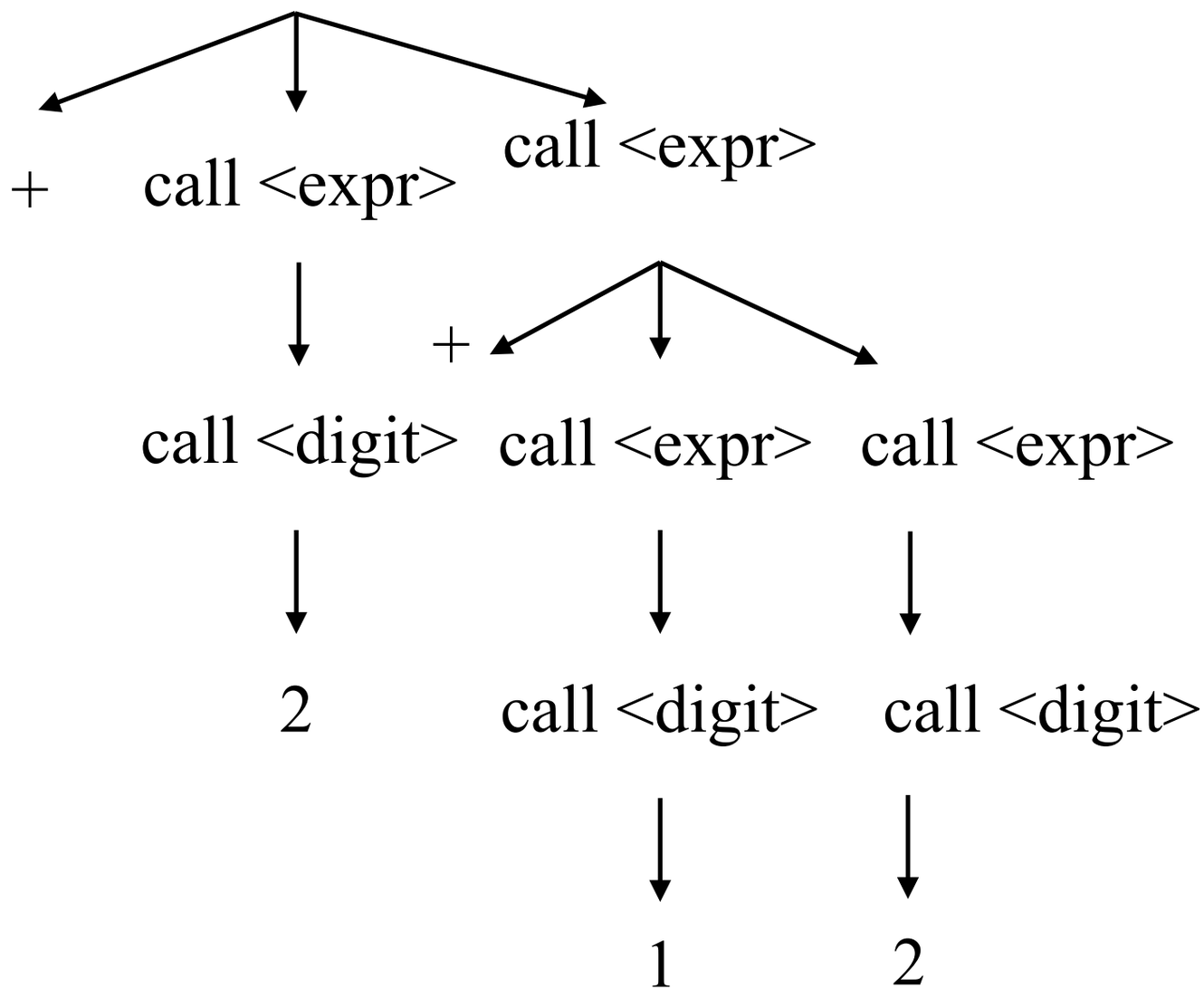


What happens when you parse expression
“ + **2** + **1** **2**”

Example: the Original Parser

1: < expr > ::= + < expr > < expr > |
2: < digit >
3: < digit > ::= 0 | 1 | 2 | 3 | … | 9

	+	0…9	other
< expr >	rule 1	rule 2	error
< digit >	error	rule 3	error

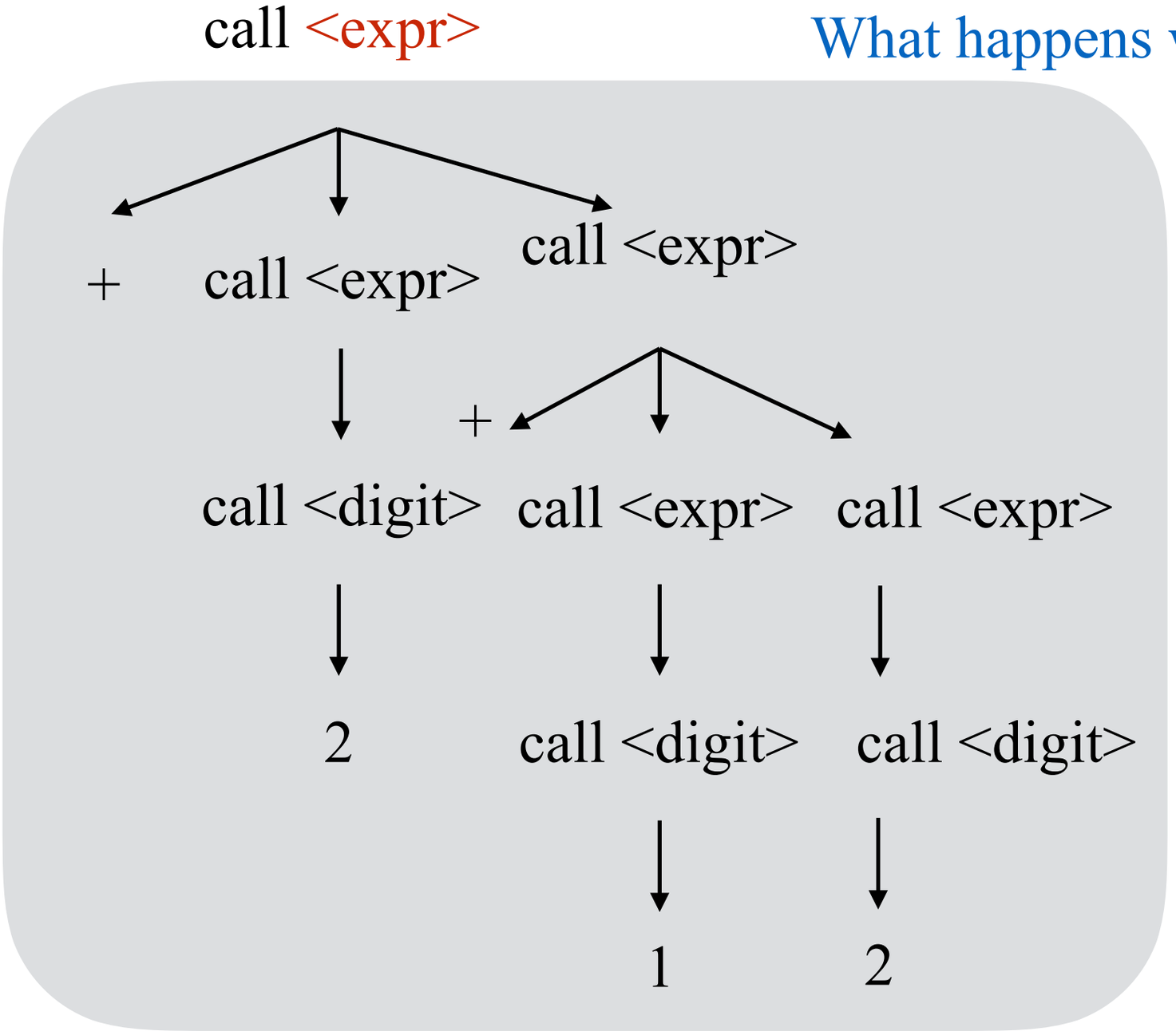


What happens when you parse expression
“ + **2** + **1 2**”

Example: the Original Parser

1: < expr > ::= + < expr > < expr > |
2: < digit >
3: < digit > ::= 0 | 1 | 2 | 3 | … | 9

	+	0…9	other
< expr >	rule 1	rule 2	error
< digit >	error	rule 3	error

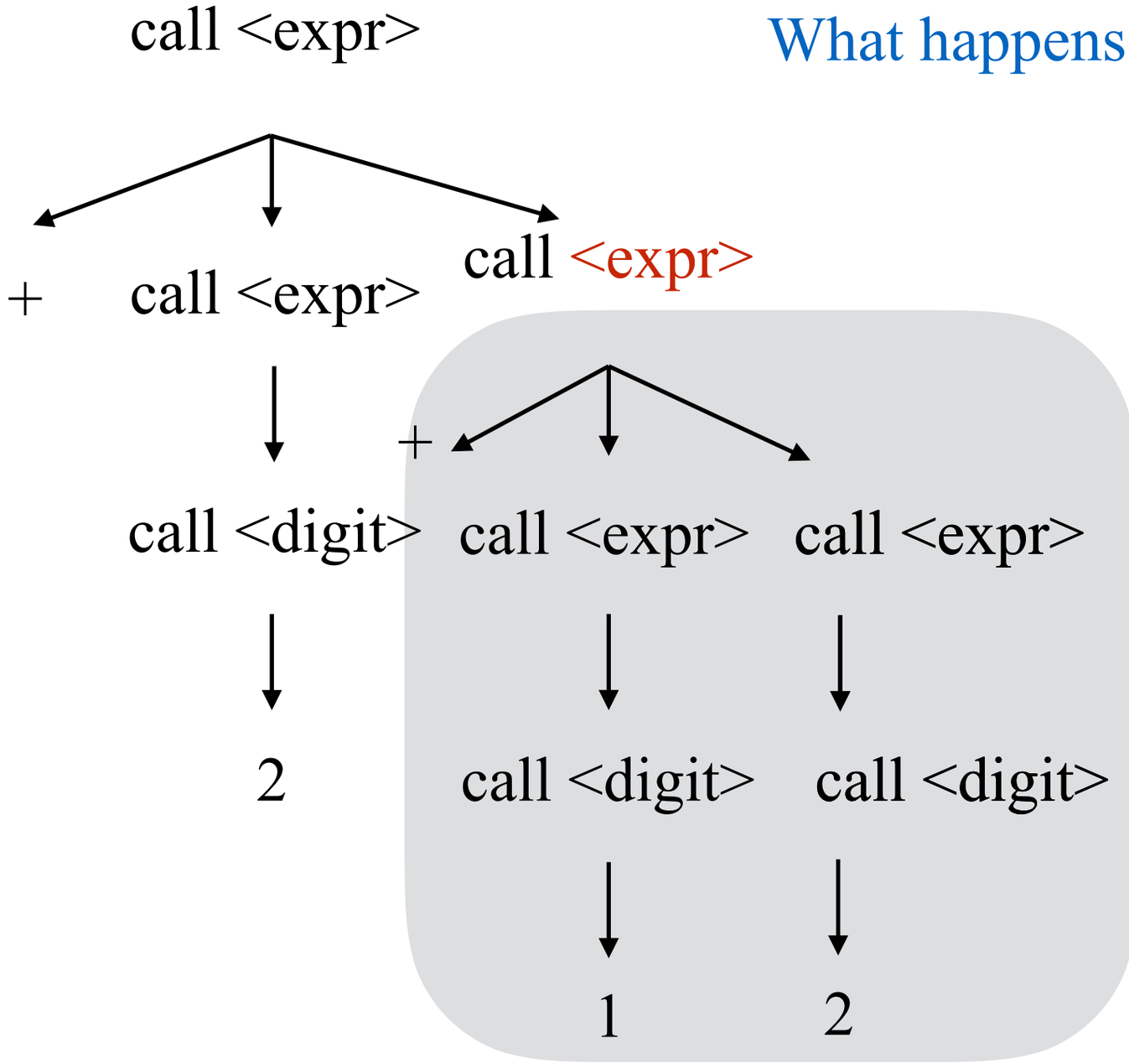


What happens when you parse expression
“ + **2** + **1** **2**”

Example: the Original Parser

1: < expr > ::= + < expr > < expr > |
2: < digit >
3: < digit > ::= 0 | 1 | 2 | 3 | … | 9

	+	0…9	other
< expr >	rule 1	rule 2	error
< digit >	error	rule 3	error



What happens when you parse expression
“ + **2** + **1** **2**”

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