
# Top-down Parsing
# Recursive Descent & LL(1)

# Roadmap (Where are we?)


*We set out to study parsing*

- Specifying syntax
  - Context-free grammars ✓
- Top-down parsers
  - Algorithm & its problem with left recursion ✓
  - Ambiguity ✓
  - Left-recursion removal ✓
- Predictive top-down parsing today
  - The LL(1) Property
  - First and Follow sets
  - Simple recursive descent parsers
  - Table-driven LL(1) parsers

## Predictive Parsing


*Given* $A \rightarrow \alpha \mid \beta$, *the parser should be able to choose between* $\alpha$ & $\beta$

Predictive Parser is a top-down parser free of backtracking

# First Sets


*For some rhs* $\alpha \in G$, FIRST($\alpha$) is set of tokens (terminals) that appear as first symbol in some string deriving from $\alpha$

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

Goal → *SheepNoise*

*SheepNoise* → *SheepNoise* baa
| baa

For *SheepNoise*:
FIRST(*Goal*) = { *baa* }
FIRST(*SN*) = { *baa* }
FIRST(*baa*) = { *baa* }

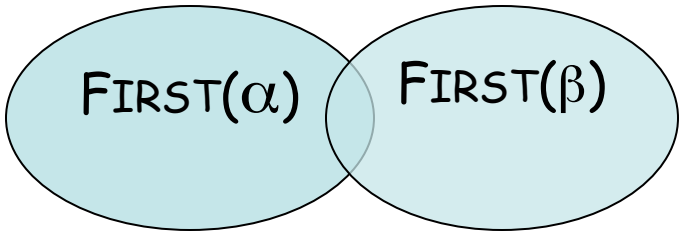
# LL(1) Property

If $A \rightarrow \alpha$ and $A \rightarrow \beta$ both appear in the grammar, we would like

$$\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \varnothing$$

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

Almost correct! See the next slide




Does not have LL(1) Property

# What about ε-productions?


If $A \rightarrow \alpha$ and $A \rightarrow \beta$ and $\varepsilon \in$ FIRST($\alpha$), then we need to ensure

$$\text{FOLLOW}(A) \cap \text{FIRST}(\beta) = \varnothing$$

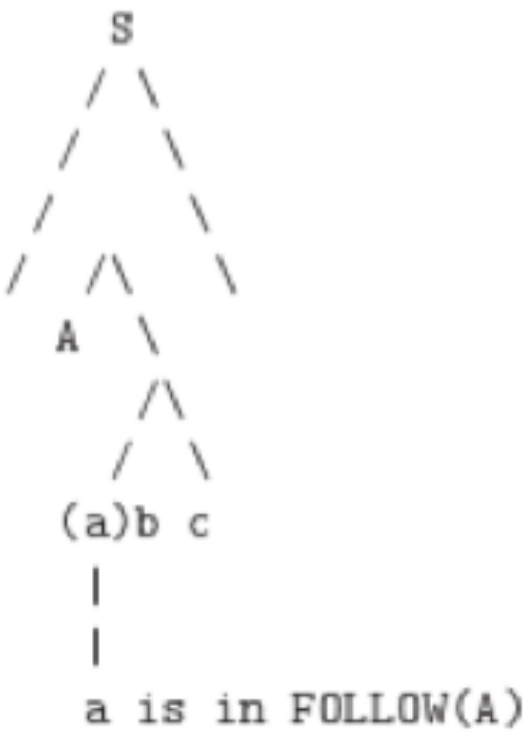
where,

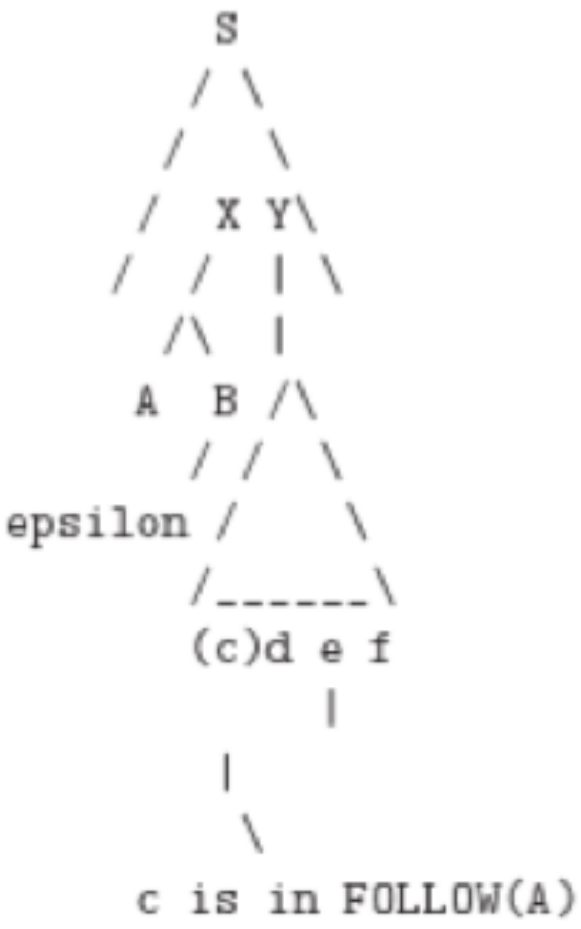
FOLLOW($A$) = the set of terminal symbols that can immediately follow $A$ in a sentential form

Formally,

Follow($A$) = {t | (t is a terminal and $G \Rightarrow^* \alpha A \underline{t} \beta$) or (t is eof and $G \Rightarrow^* \alpha A$)}

# Follow Sets


S
A
(a)b c
a is in FOLLOW(A)


S
X Y
A B
epsilon
(c)d e f
c is in FOLLOW(A)


S
(A)
EOF is in FOLLOW(A)

# FIRST$^+$ sets


## Definition of FIRST$^+(A \rightarrow \alpha)$

if $\varepsilon \in$ FIRST$(\alpha)$ then

FIRST$^+(A \rightarrow \alpha)$ = FIRST$(\alpha) \cup$ FOLLOW$(A)$

else

FIRST$^+(A \rightarrow \alpha)$ = FIRST$(\alpha)$

Grammar is *LL(1)* iff $A \rightarrow \alpha$ and $A \rightarrow \beta$ implies

$$\text{FIRST}^+(A \rightarrow \alpha) \cap \text{FIRST}^+(A \rightarrow \beta) = \varnothing$$

# What If My Grammar Is Not LL(1) ?


Can we transform a non-LL(1) grammar into an LL(1) grammar?

- In general, the answer is no
- In some cases, however, the answer is yes

- Perform:
  - —Eliminate left-recursion Friday
  - —Perform left factoring today

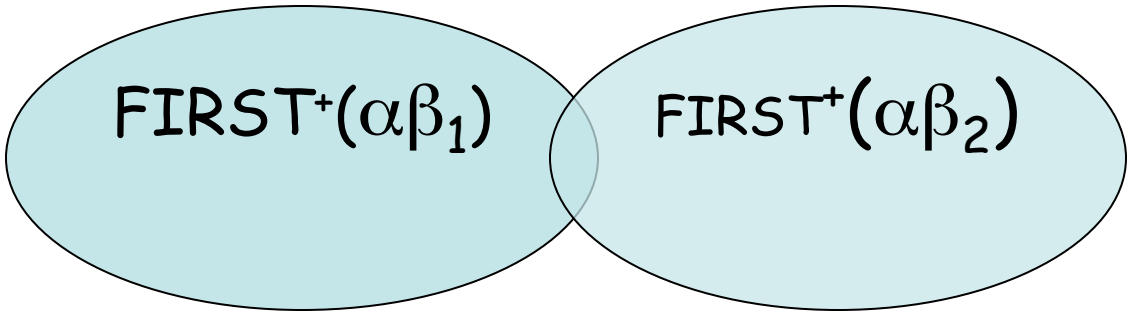
# What If My Grammar Is Not LL(1) ?

Given grammar $G$ with productions $A \rightarrow \alpha\,\beta_1$

and $A \rightarrow \alpha\,\beta_2$

if $\alpha$ derives anything other than $\varepsilon$ and

$$\text{FIRST}^{+}(A \rightarrow \alpha\,\beta_1) \cap \text{FIRST}^{+}(A \rightarrow \alpha\,\beta_2) \neq \varnothing$$




This grammar is not LL(1)

# Left Factoring

If we pull the common prefix, $\alpha$, into a separate production, we may make the grammar LL(1).

$$A \rightarrow \alpha\ A'$$

$$A' \rightarrow \beta_1$$

$$\mid\ \beta_2$$

Create a new Nonterminal

Now, if $\text{FIRST}^+(A' \rightarrow \beta_1) \cap \text{FIRST}^+(A' \rightarrow \beta_2) = \varnothing$, $G$ may be LL(1)

# Left Factoring

For each nonterminal $A$

find the longest prefix $\alpha$ common to 2 or more alternatives for $A$

if $\alpha \neq \varepsilon$ then

replace all of the productions

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \alpha\beta_3 \mid \ldots \mid \alpha\beta_n \mid \gamma$

with

$A \rightarrow \alpha A' \mid \gamma$

$A' \rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots \mid \beta_n$

Repeat until no NT has rhs' with a common prefix

NT with common prefix

# Left Factoring

For each nonterminal $A$
find the longest prefix $\alpha$ common to 2 or more alternatives for $A$
if $\alpha \neq \varepsilon$ then
replace all of the productions
$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \alpha\beta_3 \mid \ldots \mid \alpha\beta_n \mid \gamma$
with
$A \rightarrow \alpha A' \mid \gamma$
$A' \rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots \mid \beta_n$
Repeat until no NT has rhs' with a common prefix

Put common prefix $\alpha$ into a separate production rule

# Left Factoring

For each nonterminal $A$

find the longest prefix $\alpha$ common to 2 or more alternatives for $A$

if $\alpha \neq \varepsilon$ then

replace all of the productions

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \alpha\beta_3 \mid \ldots \mid \alpha\beta_n \mid \gamma$

with

$A \rightarrow \alpha A' \mid \gamma$

$A' \rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots \mid \beta_n$

Repeat until no NT has rhs' with a common prefix

Create new Nonterminal ($A'$) with all unique suffixes

# Left Factoring

For each nonterminal $A$
    find the longest prefix $\alpha$ common to 2 or more alternatives for $A$
    if $\alpha \neq \varepsilon$ then
        replace all of the productions
        $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \alpha\beta_3 \mid \ldots \mid \alpha\beta_n \mid \gamma$
        with
        $A \rightarrow \alpha A' \mid \gamma$
        $A' \rightarrow \beta_1 \mid \beta_2 \mid \beta_3 \mid \ldots \mid \beta_n$
Repeat until no NT has rhs' with a common prefix

Transformation makes some grammars into LL(1) grammars

There are languages for which no LL(1) grammar exists

# Left Factoring Example

Consider a simple right-recursive expression grammar

| | | | |
|---|---|---|---|
| 0 | *Goal* | → | *Expr* |
| 1 | *Expr* | → | *Term* + *Expr* |
| 2 | | \| | *Term* - *Expr* |
| 3 | | \| | *Term* |
| 4 | *Term* | → | *Factor* * *Term* |
| 5 | | \| | *Factor* / *Term* |
| 6 | | \| | *Factor* |
| 7 | *Factor* | → | number |
| 8 | | \| | id |

To choose between 1, 2, & 3, an LL(1) parser must look past the number or id to see the operator.

$\text{FIRST}^+(1) = \text{FIRST}^+(2) = \text{FIRST}^+(3)$

and

$\text{FIRST}^+(4) = \text{FIRST}^+(5) = \text{FIRST}^+(6)$

Let's left factor this grammar.

# Left Factoring Example

After Left Factoring, we have

| | | | |
|---|---|---|---|
| 0 | *Goal* | → | *Expr* |
| 1 | *Expr* | → | *Term Expr'* |
| 2 | *Expr'* | → | + *Expr* |
| 3 | | \| | - *Expr* |
| 4 | | \| | ε |
| 5 | *Term* | → | *Factor Term'* |
| 6 | *Term'* | → | * *Term* |
| 7 | | \| | / Term |
| 8 | | \| | ε |
| 9 | *Factor* | → | number |
| 10 | | \| | id |

Clearly,

$FIRST^+(2)$, $FIRST^+(3)$, & $FIRST^+(4)$

are disjoint, as are

$FIRST^+(6)$, $FIRST^+(7)$, & $FIRST^+(8)$

The grammar now has the LL(1) property

## FIRST Sets

FIRST($\alpha$)

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

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

# Computing FIRST Sets

```
for each x ∈ T, FIRST(x) ← {x}
for each A ∈ NT, FIRST(A) ← Ø

while (FIRST sets are still changing) do
   for each p ∈ P, of the form A→β do
      if β is B1B2…Bk then begin;
        rhs ← FIRST(B1) - {ε}
        for i ← 1 to k-1 by 1 while ε ∈ FIRST(Bi) do
             rhs ← rhs ∪ (FIRST(Bi+1) - {ε})
           end  // for loop
       end         // if-then
       if  i = k and ε ∈ FIRST(Bk)
          then rhs ← rhs ∪ {ε}
       FIRST(A) ← FIRST(A) ∪ rhs
       end  // for loop
   end      // while loop
```

Outer loop is monotone increasing for FIRST sets

→ | T ∪ NT ∪ ε | is bounded, so it terminates

Inner loop is bounded by the length of the productions in the grammar

Set empty set for terminals

# Computing FIRST Sets

```
for each x ∈ T, FIRST(x) ← {x}
for each A ∈ NT, FIRST(A) ← Ø

while (FIRST sets are still changing) do
   for each p ∈ P, of the form A→β do
      if β is B1B2…Bk then begin;
        rhs ← FIRST(B1) - {ε}
        for i ← 1 to k-1 by 1 while ε ∈ FIRST(Bi) do
             rhs ← rhs ∪ (FIRST(Bi+1) - {ε})
           end  // for loop
       end         // if-then
       if  i = k and ε ∈ FIRST(Bk)
          then rhs ← rhs ∪ {ε}
       FIRST(A) ← FIRST(A) ∪ rhs
       end  // for loop
    end      // while loop
```

Outer loop is monotone increasing for FIRST sets

→ | T ∪ NT ∪ ε | is bounded, so it terminates

Inner loop is bounded by the length of the productions in the grammar

Set empty set for First of nonterminals

# Computing FIRST Sets

```
for each x ∈ T, FIRST(x) ← {x}
for each A ∈ NT, FIRST(A) ← Ø

while (FIRST sets are still changing) do
   for each p ∈ P, of the form A→β do
      if β is B1B2…Bk then begin;
         rhs ← FIRST(B1) - {ε}
         for i ← 1 to k-1 by 1 while ε ∈ FIRST(Bi) do
               rhs ← rhs ∪ (FIRST(Bi+1) - {ε})
            end  // for loop
       end         // if-then
       if  i = k and ε ∈ FIRST(Bk)
          then rhs ← rhs ∪ {ε}
       FIRST(A) ← FIRST(A) ∪ rhs
       end  // for loop
   end      // while loop
```

Outer loop is monotone increasing for FIRST sets

→ | T ∪ NT ∪ ε | is bounded, so it terminates

Inner loop is bounded by the length of the productions in the grammar

Fixed point algorithm; Monotone because we always add to First sets

# Computing FIRST Sets

```
for each x ∈ T, FIRST(x) ← {x}
for each A ∈ NT, FIRST(A) ← Ø

while (FIRST sets are still changing) do
   for each p ∈ P, of the form A→β do
      if β is B1B2…Bk then begin;
         rhs ← FIRST(B1) - {ε}
         for i ← 1 to k-1 by 1 while ε ∈ FIRST(Bi) do
               rhs ← rhs ∪ (FIRST(Bi+1) - {ε})
            end  // for loop
      end         // if-then
      if  i = k and ε ∈ FIRST(Bk)
         then rhs ← rhs ∪ {ε}
      FIRST(A) ← FIRST(A) ∪ rhs
      end  // for loop
   end      // while loop
```

Outer loop is monotone increasing for FIRST sets

→ | T ∪ NT ∪ ε | is bounded, so it terminates

Inner loop is bounded by the length of the productions in the grammar

Iterate through each production

# Computing FIRST Sets

```
for each x ∈ T, FIRST(x) ← {x}
for each A ∈ NT, FIRST(A) ← Ø

while (FIRST sets are still changing) do
   for each p ∈ P, of the form A→β do
      if β is B1B2…Bk then begin;
        rhs ← FIRST(B1) - {ε}
        for i ← 1 to k-1 by 1 while ε ∈ FIRST(Bi) do
             rhs ← rhs ∪ (FIRST(Bi+1) - {ε})
           end  // for loop
       end         // if-then
       if  i = k and ε ∈ FIRST(Bk)
          then rhs ← rhs ∪ {ε}
       FIRST(A) ← FIRST(A) ∪ rhs
       end  // for loop
    end      // while loop
```

Outer loop is monotone increasing for FIRST sets

→ | T ∪ NT ∪ ε | is bounded, so it terminates

Inner loop is bounded by the length of the productions in the grammar

RHS is some set of T and NT.

# Computing FIRST Sets

```
for each x ∈ T, FIRST(x) ← {x}
for each A ∈ NT, FIRST(A) ← Ø

while (FIRST sets are still changing) do
   for each p ∈ P, of the form A→β do
      if β is B1B2…Bk then begin;
        rhs ← FIRST(B1) - {ε}
        for i ← 1 to k-1 by 1 while ε ∈ FIRST(Bi) do
             rhs ← rhs ∪ (FIRST(Bi+1) - {ε})
           end  // for loop
       end         // if-then
       if  i = k and ε ∈ FIRST(Bk)
          then rhs ← rhs ∪ {ε}
       FIRST(A) ← FIRST(A) ∪ rhs
       end  // for loop
   end      // while loop
```

Outer loop is monotone increasing for FIRST sets

→ | T ∪ NT ∪ ε | is bounded, so it terminates

Inner loop is bounded by the length of the productions in the grammar

Initialize rhs to First of first symbol minus epsilon

# Computing FIRST Sets

```
for each x ∈ T, FIRST(x) ← {x}
for each A ∈ NT, FIRST(A) ← Ø

while (FIRST sets are still changing) do
   for each p ∈ P, of the form A→β do
      if β is B1B2…Bk then begin;
        rhs ← FIRST(B1) - {ε}
        for i ← 1 to k-1 by 1 while ε ∈ FIRST(Bi) do
             rhs ← rhs ∪ (FIRST(Bi+1) - {ε})
           end  // for loop
       end         // if-then
       if  i = k and ε ∈ FIRST(Bk)
          then rhs ← rhs ∪ {ε}
       FIRST(A) ← FIRST(A) ∪ rhs
       end  // for loop
    end      // while loop
```

Outer loop is monotone increasing for FIRST sets

→ | T ∪ NT ∪ ε | is bounded, so it terminates

Inner loop is bounded by the length of the productions in the grammar

Iterate through symbols in production until have a symbol that does not have epsilon in First set

# Expression Grammar

| | | | |
|---|---|---|---|
| 0 | *Goal* | → | *Expr* |
| 1 | *Expr* | → | *Term Expr'* |
| 2 | *Expr'* | → | + *Term Expr'* |
| 3 | | \| | - *Term Expr'* |
| 4 | | \| | ε |
| 5 | *Term* | → | *Factor Term'* |
| 6 | *Term'* | → | * *Factor Term'* |
| 7 | | \| | / *Factor Term'* |
| 8 | | \| | ε |
| 9 | *Factor* | → | number |
| 10 | | \| | id |
| 11 | | \| | ( *Expr* ) |

| Symbol | FIRST |
|---|---|
| num | num |
| id | id |
| + | + |
| - | - |
| * | * |
| / | / |
| ( | ( |
| ) | ) |
| eof | eof |
| ε | ε |
| *Goal* | num, id, ( |
| *Expr* | num, id, ( |
| *Expr'* | +, -, ε |
| *Term* | num, id, ( |
| *Term'* | *, /, ε |
| *Factor* | num, id, ( |