Top-down Parsing
Recursive Descent & LL(1)

Roadmap (Where are we?)

- Predictive top-down parsing
  - The LL(1) Property
  - First and Follow sets
  - Simple recursive descent parsers
  - Table-driven LL(1) parsers

LL(1) Parser

- L = scan input left to right
- L = Leftmost derivation
- 1 = lookahead is enough to pick right production rule to use
- No Backtracking
- No Left Recursion

Predictive Parsing

Given production rules

$$A \rightarrow \alpha$$

$$A \rightarrow \beta$$

the parser should be able to choose between α *or* β using one lookahead

Predictive Parser is a top-down parser free of backtracking

First Sets

For some rhs $\alpha \in G$

FIRST(α) is set of tokens (terminals) that appear as first symbol in some string deriving from α

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

Some number of derivations gets us x at the beginning

Goal → *SheepNoise*

SheepNoise → *SheepNoise* baa

| baa

For *SheepNoise:*

FIRST(*Goal*) = { *baa* }

FIRST(*SN*) = { *baa* }

FIRST(*baa*) = { *baa* }

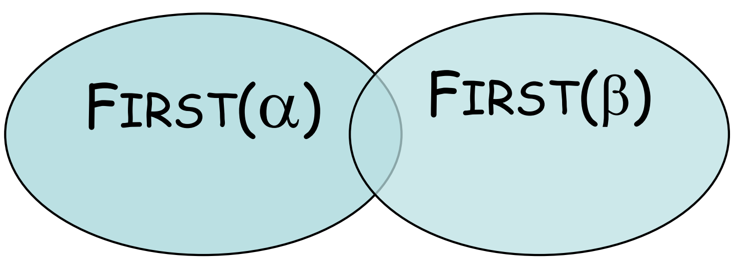
LL(1) Property

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

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

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(α), 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 $G \Rightarrow^* \alpha A$)}

Note: eof if A is at the end of the derived sentence

Follow Sets Intuition

FIRST$^+$ sets

Definition of FIRST$^+(A \to \alpha)$

if $\varepsilon \in$ FIRST(α) then

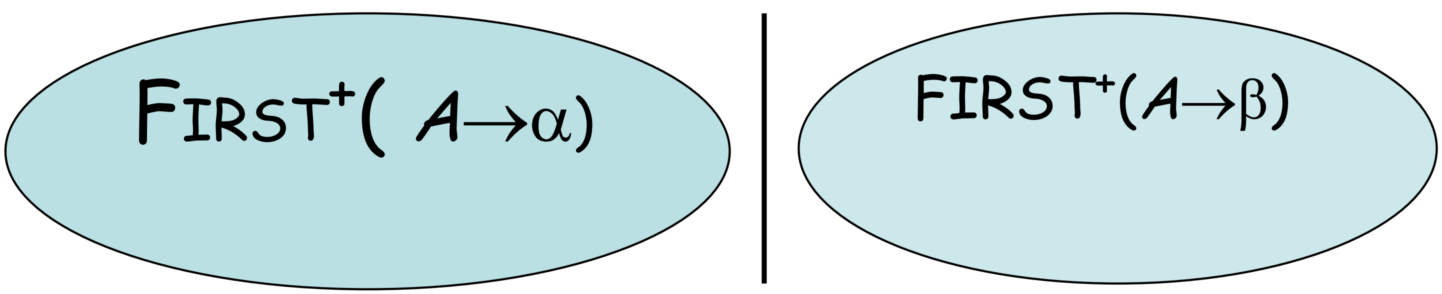
FIRST$^+(A \to \alpha)$ = FIRST$(\alpha) \cup$ FOLLOW(A)

else

FIRST$^+(A \to \alpha)$ = FIRST(α)

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

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



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 Previously
  - Perform left factoring today

What If My Grammar Is Not LL(1) ?

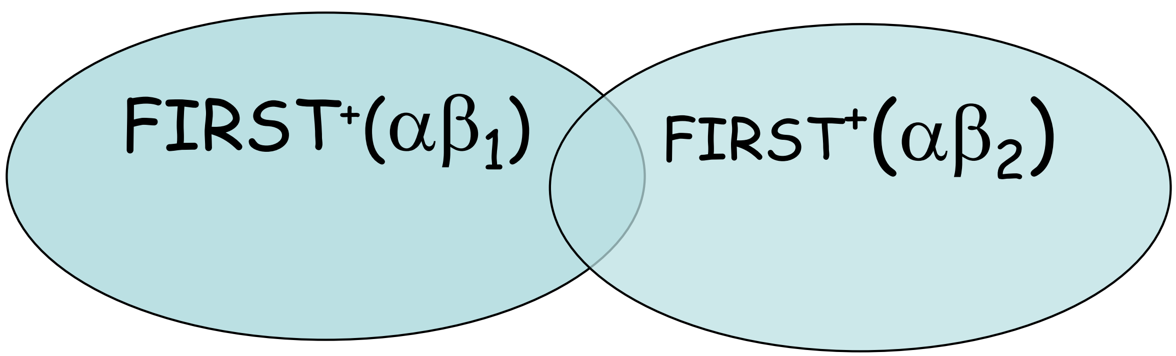
Given grammar *G* with productions

$$A \rightarrow \alpha\, \beta_1$$

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

if α derives anything other than ε and

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



This grammar is not LL(1)

Left Factoring

If we pull the common prefix, α, 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 $FIRST^+(A' \rightarrow \beta_1) \cap FIRST^+(A' \rightarrow \beta_2) = \varnothing$, *G* may be LL(1)

Left Factoring

For each nonterminal A

find the longest prefix α 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 α 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 α into a separate production rule

Left Factoring

For each nonterminal A

find the longest prefix α common to 2 or more alternatives for A

if $\alpha \neq \varepsilon$ then

replace all of the productions

$A \rightarrow \alpha\, \beta_1 \mid \alpha\, \beta_2 \mid \alpha\, \beta_3 \mid \dots \mid \alpha\, \beta_n \mid \gamma$

with

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

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

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 α 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

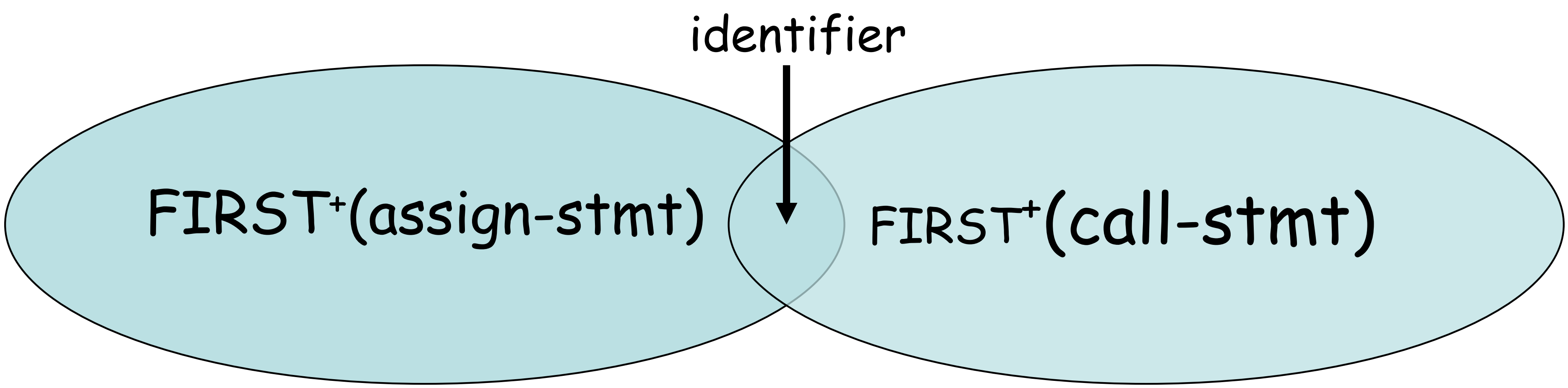
Transformation makes some grammars into LL(1) grammars

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

Left Factoring not possible

Here is an example where a programming language fails to be LL(1) and is not in a form that can be left factored

statement $\longrightarrow$ *assign-stmt* | *call-stmt* | **other**
assign-stmt $\longrightarrow$ **identifier** := *exp*
call-stmt $\longrightarrow$ **identifier** (*exp-list*)



Left Factoring Example

Consider a simple right-recursive expression grammar

0	*Goal*	$\rightarrow$	*Expr*
1	*Expr*	$\rightarrow$	*Term* + *Expr*
2		|	*Term* - *Expr*
3		|	*Term*
4	*Term*	$\rightarrow$	*Factor* * *Term*
5		|	*Factor* / *Term*
6		|	*Factor*
7	*Factor*	$\rightarrow$	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*	$\rightarrow$	*Expr*
1	*Expr*	$\rightarrow$	*Term Expr'*
2	*Expr'*	$\rightarrow$	+ *Expr*
3		|	- *Expr*
4		|	ε
5	*Term*	$\rightarrow$	*Factor Term'*
6	*Term'*	$\rightarrow$	* *Term*
7		|	/ Term
8		|	ε
9	*Factor*	$\rightarrow$	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(α)

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

That is, <u>x</u> $\in$ FIRST(α) *iff* $\alpha \Rightarrow^* \underline{x}\,\gamma$, for some γ

Computing FIRST Sets

for each $x \in T$*, FIRST(x)* $\leftarrow \{x\}$
for each $A \in NT$*, FIRST(A)* $\leftarrow \emptyset$

while (FIRST sets are still changing) do
for each $p \in P$*, of the form* $A \rightarrow \beta$ *do*
if β *is* $B_1 B_2 \ldots B_k$ *then begin;*
$FS \leftarrow FIRST(B_1) - \{\varepsilon\}$
for $i \leftarrow 1$ *to* $k\text{-}1$ *by 1 while* $\varepsilon \in FIRST(B_i)$ *do*
$FS \leftarrow FS \cup (FIRST(B_{i+1}) - \{\varepsilon\})$
end // for loop
end // if-then
if $i = k$ *and* $\varepsilon \in FIRST(B_k)$
then $FS \leftarrow FS \cup \{\varepsilon\}$
$FIRST(A) \leftarrow FIRST(A) \cup FS$
end // for loop
end // while loop

Outer loop is monotone increasing for FIRST sets

$\rightarrow |T \cup NT \cup \varepsilon|$ is bounded, so it terminates

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

Set 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;
      FS ← FIRST(B1) - {ε}
      for i ← 1 to k-1 by 1 while ε ∈ FIRST(Bi) do
          FS ← FS ∪ (FIRST(Bi+1) - {ε})
        end  // for loop
    end         // if-then
    if  i = k and ε ∈ FIRST(Bk)
       then FS ← FS ∪ {ε}
    FIRST(A) ← FIRST(A) ∪ FS
    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;
            FS ← FIRST(B1) - {ε}
            for i ← 1 to k-1 by 1 while ε ∈ FIRST(Bi) do
                FS ← FS ∪ (FIRST(Bi+1) - { ε })
            end  // for loop
        end          // if-then
        if  i = k and ε ∈ FIRST(Bk)
            then FS ← FS ∪ {ε}
        FIRST(A) ← FIRST(A) ∪ FS
    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; never delete from 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;
       FS ← FIRST(B1) – {ε}
       for i ← 1 to k–1 by 1 while ε ∈ FIRST(Bi) do
           FS ← FS ∪ (FIRST(Bi+1) – { ε } )
         end  // for loop
     end          // if-then
     if  i = k and ε ∈ FIRST(Bk )
        then FS ← FS ∪ {ε}
     FIRST(A) ← FIRST(A) ∪ FS
     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 $x \in T$, $FIRST(x) \leftarrow \{x\}$
for each $A \in NT$, $FIRST(A) \leftarrow \emptyset$

while (FIRST sets are still changing) do
    for each $p \in P$, *of the form* $A \rightarrow \beta$ *do*
        if β *is* $B_1B_2 \ldots B_k$ *then begin;*
            $FS \leftarrow FIRST(B_1) - \{\varepsilon\}$
            for $i \leftarrow 1$ *to* $k\text{-}1$ *by 1 while* $\varepsilon \in FIRST(B_i)$ *do*
                $FS \leftarrow FS \cup (FIRST(B_{i+1}) - \{\varepsilon\})$
            end // for loop
        end // if-then
        if $i = k$ *and* $\varepsilon \in FIRST(B_k)$
            then $FS \leftarrow FS \cup \{\varepsilon\}$
        $FIRST(A) \leftarrow FIRST(A) \cup FS$
    end // for loop
end // while loop

Outer loop is monotone increasing for FIRST sets

$\rightarrow |T \cup NT \cup \varepsilon|$ 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;
            FS ← FIRST(B1) - {ε}
            for i ← 1 to k-1 by 1 while ε ∈ FIRST(Bi) do
                FS ← FS ∪ (FIRST(Bi+1) - { ε })
            end  // for loop
        end       // if-then
        if  i = k and ε ∈ FIRST(Bk)
            then FS ← FS ∪ {ε}
        FIRST(A) ← FIRST(A) ∪ FS
        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;
            FS ← FIRST(B1) - {ε}
            for i ← 1 to k-1 by 1 while ε ∈ FIRST(Bi) do
                FS ← FS ∪ (FIRST(Bi+1) - { ε })
            end  // for loop
        end          // if-then
        if  i = k and ε ∈ FIRST(Bk)
            then FS ← FS ∪ {ε}
        FIRST(A) ← FIRST(A) ∪ FS
        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*	$\rightarrow$	*Expr*
1	*Expr*	$\rightarrow$	*Term Expr'*
2	*Expr'*	$\rightarrow$	+ *Term Expr'*
3		|	- *Term Expr'*
4		|	ε
5	*Term*	$\rightarrow$	*Factor Term'*
6	*Term'*	$\rightarrow$	* *Factor Term'*
7		|	/ *Factor Term'*
8		|	ε
9	*Factor*	$\rightarrow$	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, (