Building a Predictive Parser

I.e., How to build the parse table for a recursive-descent parser

Last Time: Intro LL(1) Predictive Parser

Predict the parse tree top-down

Parser structure

- 1 token of lookahead
- A stack tracking the current parse tree's frontier
- Selector/parse table

Necessary conditions

- Left-factored
- Free of left-recursion

Today: Building the Parse Table

Review grammar transformations

- Why they are necessary
- How they work

Build the parse table

- FIRST(*X*): Set of terminals that can begin at a subtree rooted at *X*
- FOLLOW(*X*): Set of terminals that can appear after *X*

Review of LL(1) Grammar Transformations

Necessary (but not sufficient conditions) for LL(1) parsing:

- Free of left recursion
  - "No left-recursive rules"
  - Why? Need to look past the list to know when to cap it
- Left-factored
  - "No rules with a common prefix, for any nonterminal"
  - Why? We would need to look past the prefix to pick the production

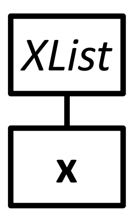
Why Left Recursion is a Problem (Blackbox View)

CFG snippet: $\mathit{XList} \longrightarrow \mathit{XList}\ \mathbf{x} \mid \mathbf{x}$

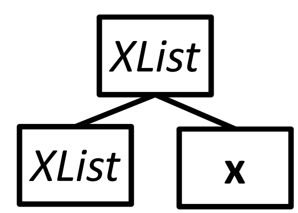
Current parse tree: $\boxed{\mathit{XList}}$

Current token: **x**

How should we grow the tree top-down?



(OR)



Correct if there are no more **x**s

Correct if there <u>are</u> more **x**s

We don’t know which to choose without more lookahead

Why Left Recursion is a Problem (Whitebox View)

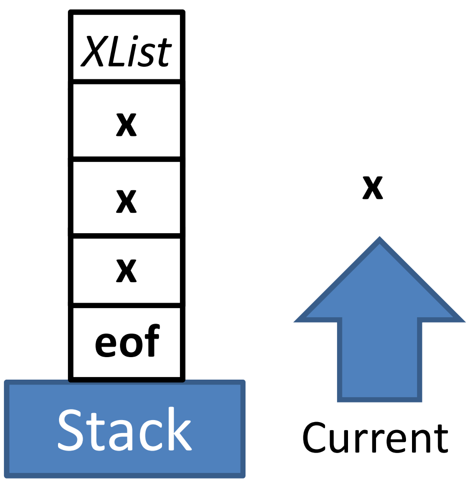
CFG snippet: $XList \longrightarrow XList\ \mathbf{x} \mid \mathbf{x}$

Current parse tree: $\boxed{XList}$

Current token: **x**

Parse table:

	x	**eof**
XList	*XList* **x**	ε



(Stack overflow)

Left-Recursion Elimination: Review

Replace $A \longrightarrow A\,\alpha \mid \beta$

Head of the list

With $A \longrightarrow \beta\, A'$

$A' \longrightarrow \alpha\, A' \mid \varepsilon$

Where β does not start with *A, or* may not be present

Preserves the language (a list of αs, starting with a β), but uses right recursion

Left-Recursion Elimination: Ex1

$$A \longrightarrow A\,\alpha \mid \beta \quad \Rightarrow \quad \begin{aligned} A &\longrightarrow \beta\, A' \\ A' &\longrightarrow \alpha\, A' \mid \varepsilon \end{aligned}$$

$$E \longrightarrow E\,\underbrace{\textbf{cross id}}_{\alpha} \mid \underbrace{\textbf{id}}_{\beta} \quad \Rightarrow \quad \begin{aligned} E &\longrightarrow \overbrace{\textbf{id}}^{\beta}\, E' \\ E' &\longrightarrow \underbrace{\textbf{cross id}}_{\alpha}\, E' \mid \varepsilon \end{aligned}$$

Left-Recursion Elimination: Ex2

$$A \longrightarrow A\,\alpha \mid \beta \quad \Rightarrow \quad \begin{aligned} A &\longrightarrow \beta\, A' \\ A' &\longrightarrow \alpha\, A' \mid \varepsilon \end{aligned}$$

$$\begin{aligned} E &\longrightarrow E\ \mathbf{+}\ T \mid T \\ T &\longrightarrow T\ \mathbf{*}\ F \mid F \\ F &\longrightarrow \mathbf{(}\ \mathrm{E}\ \mathbf{)} \mid \mathbf{id} \end{aligned} \quad \Rightarrow \quad \begin{aligned} E &\longrightarrow T\, E' \\ E' &\longrightarrow \mathbf{+}\ \mathrm{T\ E'} \mid \varepsilon \\ T &\longrightarrow F\, T' \\ T' &\longrightarrow \mathbf{*}\ \mathrm{F\ T'} \mid \varepsilon \\ F &\longrightarrow \mathbf{(}\ \mathrm{E}\ \mathbf{)} \mid \mathbf{id} \end{aligned}$$

Left-Recursion Elimination: Ex3

$$A \longrightarrow A\ \alpha \mid \beta \quad \Rightarrow \quad \begin{array}{l} A \longrightarrow \beta\ A' \\ A' \longrightarrow \alpha\ A' \mid \varepsilon \end{array}$$

$$\begin{array}{lcl} \mathit{DList} & \longrightarrow & \mathit{DList}\ D \mid \varepsilon \\ D & \longrightarrow & \mathit{Type}\ \textbf{id semi} \\ \mathit{Type} & \longrightarrow & \textbf{bool} \mid \textbf{int} \end{array} \quad \Rightarrow \quad \begin{array}{lcl} \mathit{DList} & \longrightarrow & \varepsilon\ \mathit{DList}' \\ \mathit{DList}' & \longrightarrow & D\ \mathit{DList}' \mid \varepsilon \\ D & \longrightarrow & \mathit{Type}\ \textbf{id semi} \\ \mathit{Type} & \longrightarrow & \textbf{bool} \mid \textbf{int} \end{array}$$

$$\begin{array}{lcl} \mathit{DList} & \longrightarrow & D\ \mathit{DList} \mid \varepsilon \\ D & \longrightarrow & \mathit{Type}\ \textbf{id semi} \\ \mathit{Type} & \longrightarrow & \textbf{bool} \mid \textbf{int} \end{array}$$

Left Factoring: Review

Removing a common prefix from a grammar

Replace $A \longrightarrow \boxed{\alpha\, \beta_1 \mid \ldots \mid \alpha\, \beta_m} \mid y_1 \mid \ldots \mid y_n$

With $A \longrightarrow \boxed{\alpha\, A'} \mid y_1 \mid \ldots \mid y_n$

$A' \longrightarrow \beta_1 \mid \ldots \mid \beta_m$

Where β_i and y_i are sequence of symbols with no common prefix

Note: y_i may not be present, and one of the β may be ε

Combine all "problematic" rules that start with α into one rule $\alpha\, A'$

Now A' represents the suffix of the "problematic" rules

Left Factoring: Example 1

$$A \longrightarrow \alpha\,\beta_1 \mid \dots \mid \alpha\,\beta_m \mid \gamma_1 \mid \dots \mid \gamma_n \quad \Rightarrow \quad \begin{aligned} A &\longrightarrow \alpha\, A' \mid \gamma_1 \mid \dots \mid \gamma_n \\ A' &\longrightarrow \beta_1 \mid \dots \mid \beta_m \end{aligned}$$

$$X \longrightarrow \overbrace{\mathbf{<}}^{\alpha}\,\overbrace{\mathbf{a >}}^{\beta_1} \mid \overbrace{\mathbf{<}}^{\alpha}\,\overbrace{\mathbf{b >}}^{\beta_2} \mid \overbrace{\mathbf{<}}^{\alpha}\,\overbrace{\mathbf{c >}}^{\beta_3} \mid \overbrace{\mathbf{d}}^{\gamma_1}$$

$$X \longrightarrow \overbrace{\mathbf{<}}^{\alpha}\, X' \mid \overbrace{\mathbf{d}}^{\gamma_1}$$

$$X' \longrightarrow \underbrace{\mathbf{a >}}_{\beta_1} \mid \underbrace{\mathbf{b >}}_{\beta_2} \mid \underbrace{\mathbf{c >}}_{\beta_3}$$

Left Factoring: Example 2

$A \longrightarrow \alpha\,\beta_1 \mid \ldots \mid \alpha\,\beta_m \mid \gamma_1 \mid \ldots \mid \gamma_n$ ➡ $A \longrightarrow \alpha\,A' \mid \gamma_1 \mid \ldots \mid \gamma_n$
$A' \longrightarrow \beta_1 \mid \ldots \mid \beta_m$

Stmt ⟶ **id** $\overbrace{\textbf{assign}\ E}^{\beta_1}$ | **id** $\overbrace{\textbf{(}\ \mathit{EList}\ \textbf{)}}^{\beta_2}$ | **return**

E ⟶ **intlit** | **id**

Elist ⟶ *E* | *E* **comma** *EList*

Stmt ⟶ **id** *Stmt’* | **return**

Stmt’ ⟶ **assign** *E* | **(** *EList* **)**

E ⟶ **intlit** | **id**

Elist ⟶ *E* | *E* **comma** *EList*

Left Factoring: Example 3

$$A \longrightarrow \alpha\,\beta_1 \mid \ldots \mid \alpha\,\beta_m \mid \gamma_1 \mid \ldots \mid \gamma_n \quad \Rightarrow \quad \begin{aligned} A &\longrightarrow \alpha\,A' \mid \gamma_1 \mid \ldots \mid \gamma_n \\ A' &\longrightarrow \beta_1 \mid \ldots \mid \beta_m \end{aligned}$$

$$S \longrightarrow \overbrace{\textbf{if } E \textbf{ then } S}^{\alpha}\,{}^{\beta_1 = \varepsilon} \mid \overbrace{\textbf{if } E \textbf{ then } S}^{\alpha}\ \overbrace{\textbf{else } S}^{\beta_2} \mid \textbf{semi}$$

$$E \longrightarrow \textbf{boollit}$$

$$S \longrightarrow \textbf{if } E \textbf{ then } S\ S' \mid \textbf{semi}$$

$$S' \longrightarrow \textbf{else } S \mid \varepsilon$$

$$E \longrightarrow \textbf{boollit}$$

Left Factoring: Not Always Immediate

$A \longrightarrow \alpha\,\beta_1 \mid \ldots \mid \alpha\,\beta_m \mid y_1 \mid \ldots \mid y_n$

$A \longrightarrow \alpha\, A' \mid y_1 \mid \ldots \mid y_n$
$A' \longrightarrow \beta_1 \mid \ldots \mid \beta_m$

This snippet yearns for left factoring

S ⟶ A | C | **return**
A ⟶ **id assign** *E*
C ⟶ **id (** *EList* **)**

but we cannot! At least without *inlining*

S ⟶ **id assign** *E* | **id (** *Elist* **)** | **return**

Let's be more constructive

So far, we have only talked about what <u>precludes</u> us from building a predictive parser

It is time to actually build the parse table

Building the Parse Table

What do we actually need to ensure that production $A \longrightarrow \alpha$ is the correct one to apply?

Assume α is an arbitrary sequence of symbols

1. What terminals could α possibly start with
   → we call this the FIRST set
2. What terminal could possibly come after A
   → we call this the FOLLOW set

Why is FIRST Important?

Assume the top-of-stack symbol is A and current token is a

- Production 1: $A \longrightarrow \alpha$
- Production 2: $A \longrightarrow \beta$

FIRST lets us disambiguate:

- If a is in FIRST(α), we know Production 1 is a viable choice
- If a is in FIRST(β), we know Production 2 is a viable choice
- If a is only in one of FIRST(α) and FIRST(β), we can predict the production we need

FIRST Sets

FIRST(α) is the set of terminals that begin the strings derivable from α, and also, if α can derive ε, then ε is in FIRST(α).

Formally, let's write it together

FIRST(α) =

FIRST Sets

FIRST(α) is the set of terminals that begin the strings derivable from α, and also, if α can derive ε, then ε is in FIRST(α).

Formally, let's write it together

FIRST(α) = $\{t | (t \in \Sigma \wedge \alpha \Rightarrow^* t\beta) \vee (t = \epsilon \wedge \alpha \Rightarrow^* \epsilon)\}$

FIRST Construction: Single Symbol

We begin by doing FIRST sets for a single, arbitrary symbol X

- If X is a terminal: FIRST(X) = { X }
- If X is ε: FIRST(ε) = { ε }
- If X is a nonterminal, for each $X \longrightarrow Y_1\ Y_2 \ldots Y_k$
  - Put FIRST(Y_1) - {ε} into FIRST(X)
  - If ε is in FIRST(Y_1) , put FIRST(Y_2) - {ε} into FIRST(X)
  - If ε is also in FIRST(Y_2), put FIRST(Y_3) - {ε} into FIRST(X)
  - …
  - If ε is in FIRST of all Y_i symbols, put ε into FIRST(X)

Repeat this step until there are no changes to any nonterminal's FIRST set

FIRST(X) Example

<u>Building FIRST(X) for nonterm X</u>

for each $X \longrightarrow Y_1\ Y_2\ \dots\ Y_k$

- Add FIRST(Y_1) - {ε}
- If ε is in FIRST($Y_{1 \text{ to } i-1}$): add FIRST(Y_i) - {ε}
- If ε is in all RHS symbols, add ε

Exp	⟶	*Term Exp'*
Exp'	⟶	**minus** *Term Exp'* | ε
Term	⟶	*Factor Term'*
Term'	⟶	**divide** *Factor Term'* | ε
Factor	⟶	**intlit** | **lparen** *Exp* **rparen**

FIRST(*Factor*) = { **intlit, lparen** }

FIRST(*Term'*) = { **divide,** ε }

FIRST(*Term*) = { **intlit, lparen** }

FIRST(*Exp'*) = { **minus,** ε}

FIRST(*Exp*) = { **intlit, lparen**}

FIRST(α)

- We now extend FIRST to strings of symbols α
  - We want to define FIRST for all RHS
- Looks very similar to the procedure for single symbols
- Let $\alpha = Y_1 \ Y_2 \ \ldots \ Y_k$
  - Put FIRST(Y_1) - {ε} in FIRST(α)
    - If ε is in FIRST(Y_1): add FIRST(Y_2) – {ε} to FIRST(α)
    - If ε is in FIRST(Y_2): add FIRST(Y_3) – {ε} to FIRST(α)
    - …
    - If ε is in FIRST of all Y_i symbols, put ε into FIRST(α)

Building FIRST(α) from FIRST(X)

<u>Building FIRST(X) for nonterm X</u>

for each $X \longrightarrow Y_1\ Y_2 \ldots Y_k$

- Add FIRST(Y_1) - {ε}
- If ε is in FIRST($Y_{1 \text{ to } i-1}$): add FIRST(Y_i) - {ε}
- If ε is in all RHS symbols, add ε

<u>Building FIRST(α)</u>

Let $\alpha = Y_1\ Y_2 \ldots Y_k$

- Add FIRST(Y_1) - {ε}
- If ε is in FIRST($Y_{1 \text{ to } i-1}$): add FIRST(Y_i) – {ε}
- If ε is in all RHS symbols, add ε

FIRST(α) Example

<u>Building FIRST(α)</u>

Let $\alpha = Y_1\ Y_2\ \ldots\ Y_k$

- Add FIRST(Y_1) - {ε}
- If ε is in FIRST($Y_{1 \text{ to } i-1}$): add FIRST(Y_i) – {ε}
- If, for all RHS symbols Y_j, ε is in FIRST(Y_j), add ε

E → T X
X → **+** T X | ε
T → F Y
Y → ***** F Y | ε
F → **(** E **)** | **id**

FIRST(*E*) = {**(**, **id**}
FIRST(*T*) = {**(**, **id**}
FIRST(*F*) = {**(**, **id**}
FIRST(*X*) = {**+**, ε}
FIRST(*Y*) = {*****, ε}

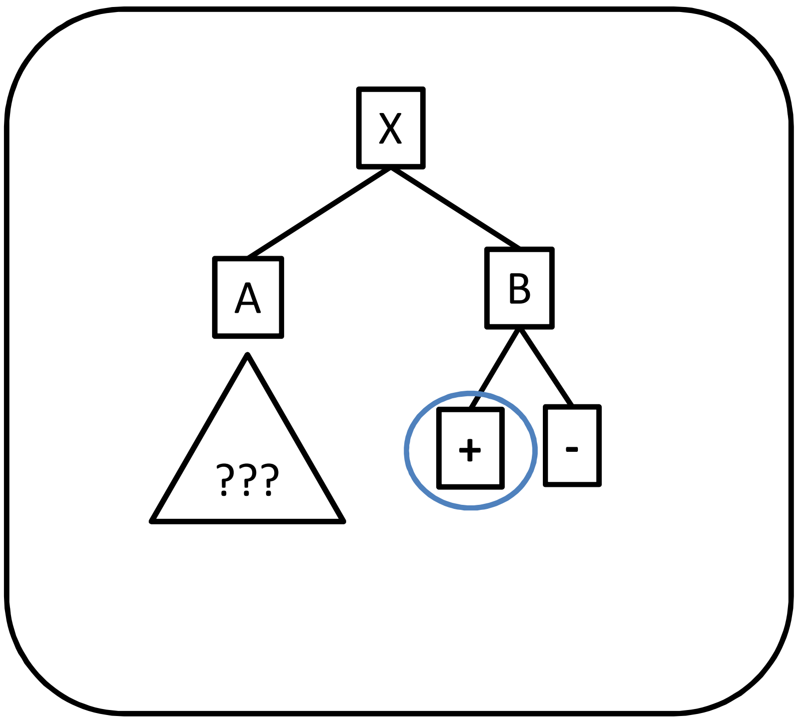
FIRST(*T X*) = {**(**, **id**}
FIRST(**+** *T X*) = { **+** }
FIRST(*F Y*) = { **(**, **id** }
FIRST (* *F Y*) = { ***** }
FIRST(**(** *E* **)**) = { **(** }
FIRST(**id**) = { **id** }

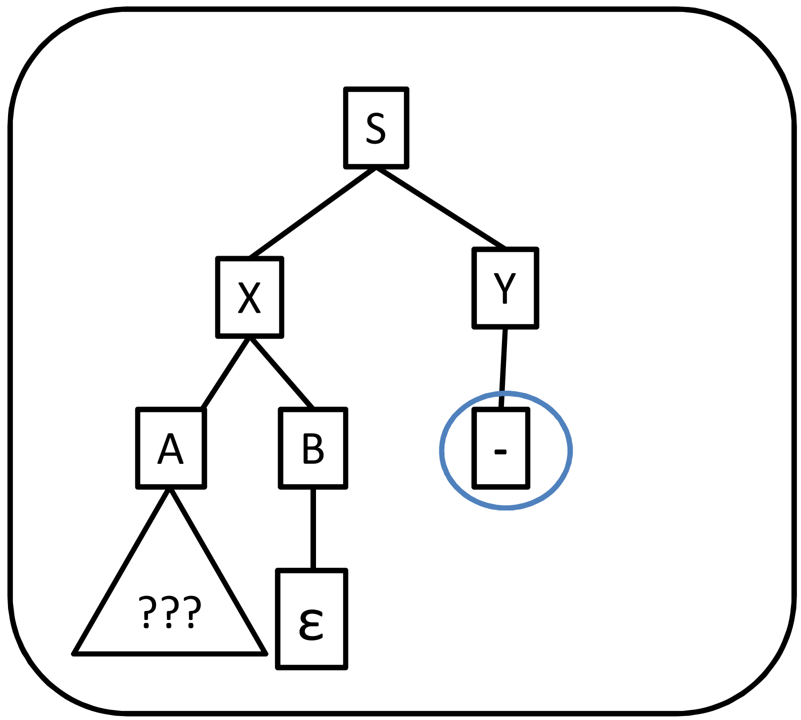
FIRST sets alone do not provide enough information to construct a parse table

If a rule R can derive ε, we need to know what terminals can come just <u>after</u> R

FOLLOW Sets: Pictorially

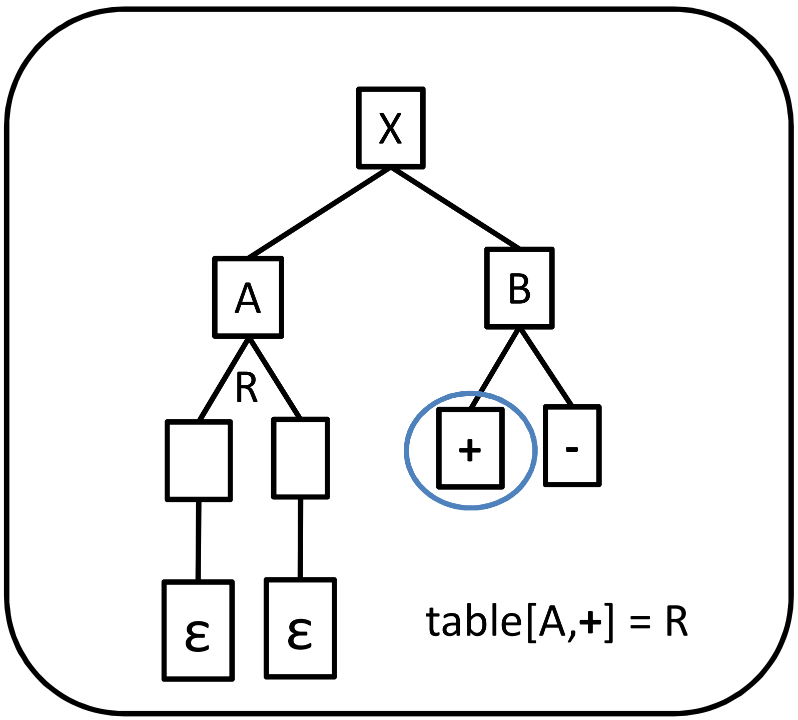
For nonterminal A, FOLLOW(A) is the set of terminals that can appear immediately to the right of A

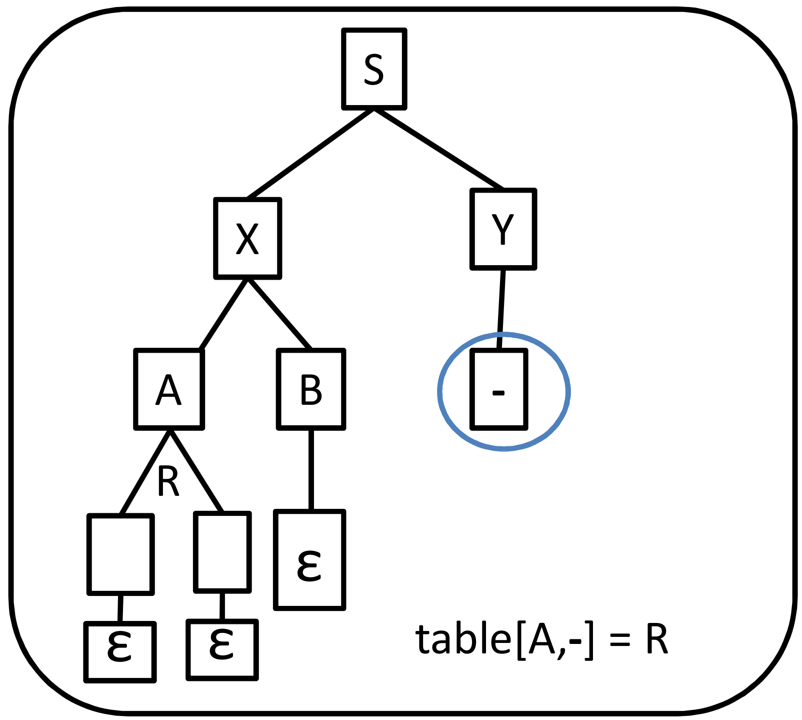




FOLLOW Sets: Pictorially

For nonterminal A, FOLLOW(A) is the set of terminals that can appear immediately to the right of A





FOLLOW Sets

For nonterminal A, FOLLOW(A) is the set of terminals that can appear immediately to the right of A

Let's write it together,

FOLLOW(A) =

FOLLOW Sets

For nonterminal A, FOLLOW(A) is the set of terminals that can appear immediately to the right of A

Let's write it together,

FOLLOW(A) =

$$\{t | (t \in \Sigma \wedge S \Rightarrow^{+} \alpha A t \beta) \vee (t = EOF \wedge S \Rightarrow^{*} \alpha A)\}$$

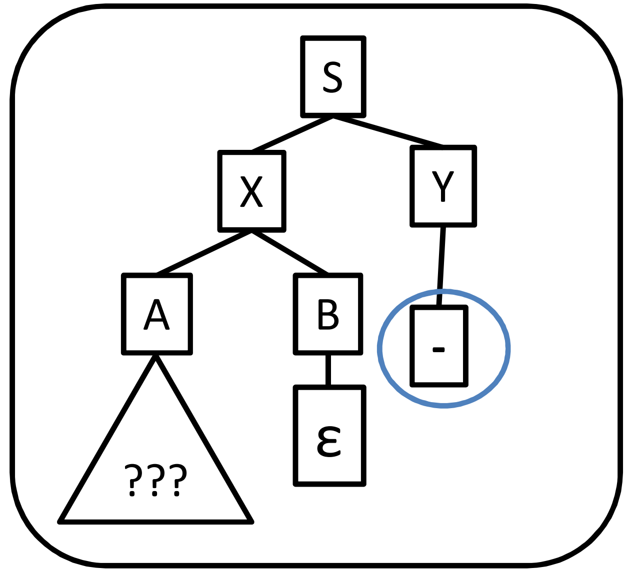
FOLLOW Sets: Construction

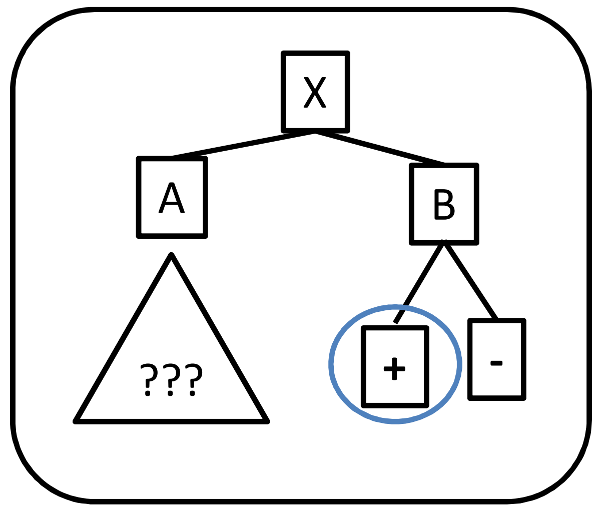
To build FOLLOW(A)

- If A is the start nonterminal, add eof
- For rules $X \longrightarrow \alpha\ A\ \beta$
  - Add FIRST(β) – {ε}
  - If ε is in FIRST(β) or β is empty, add FOLLOW(X)

Where α, β may be empty

Continue building FOLLOW sets until reach a fixed point (i.e., no more symbols can be added)





FOLLOW Sets Example

FOLLOW(A) for $X \longrightarrow \alpha\ A\ \beta$

If A is the start, add **eof**

Add FIRST(β) – {ε}

Add FOLLOW(X) if ε in FIRST(β) or β is empty

S	⟶	B **c** | D B
B	⟶	**a b** | **c** S
D	⟶	**d** | ε

FIRST (S) = { **a**, **c, d** }

FIRST (B) = { **a**, **c** }

FIRST (D) = { **d**, ε }

FIRST (B **c**) = { **a**, **c** }

FIRST (D B) = { **d**, **a**, **c** }

FIRST (**a b**) = { **a** }

FIRST (**c** *S*) = { **c** }

FOLLOW (S) = { **eof** }

FOLLOW (B) = { **c**, **eof** }

FOLLOW (D) = { **a, c** }

FOLLOW (S) = { **eof, c** }

FOLLOW (B) = { **c**, **eof** }

FOLLOW (D) = { **a, c** }

FOLLOW (S) = { **eof, c** }

FOLLOW (B) = { **c**, **eof** }

FOLLOW (D) = { **a, c** }

Building the Parse Table

```
for each production X ⟶ α {
  for each terminal t in FIRST(α){
    put α in Table[X][t]
  }
  if ε is in FIRST(α){
    for each terminal t in FOLLOW(X){
      put α in Table[X][t]
    }
  }
}
```

Table collision ⇔ Grammar is not in LL(1)

Putting it all together

Build FIRST sets for each nonterminal

Build FIRST sets for each production's RHS

Build FOLLOW sets for each nonterminal

Use FIRST and FOLLOW to fill parse table for each production

Tips n’ Tricks

FIRST sets

– Only contain alphabet terminals and ε
– Defined for arbitrary RHS and nonterminals
– Constructed by starting at the beginning of a production

FOLLOW sets

– Only contain alphabet terminals and eof
– Defined for nonterminals only
– Constructed by jumping into production

FIRST(α) for $\alpha = Y_1 Y_2 \ldots Y_k$

Add FIRST(Y_1) - {ε}

If ε is in FIRST($Y_{1 \text{ to } i-1}$): add FIRST(Y_i) – {ε}

If ε is in all RHS symbols, add ε

FOLLOW(A) for $X \longrightarrow \alpha\, A\, \beta$

If A is the start, add **eof**

Add FIRST(β) – {ε}

Add FOLLOW(*X*) if ε in FIRST(β) or β empty

Table[X][t]

```
for each production X ⟶ α
 for each terminal t in FIRST(α)
   put α in Table[X][t]
 if ε is in FIRST(α){
   for each terminal t in FOLLOW(X){
     put α in Table[X][t]
```

CFG

S ⟶ B **c** | D B

B ⟶ **a b** | **c** S

D ⟶ **d** | ε

Not LL(1)

FIRST (S) = { **a**, **c**, **d** }

FIRST (B) = { **a**, **c** }

FIRST (D) = { **d**, ε }

FIRST (B **c**) = { **a**, **c** }

FIRST (D B) = { **d**, **a**, **c** }

FIRST (**a b**) = { **a** }

FIRST (**c** *S*) = { **c** }

FIRST (**d**) = { **d** }

FIRST (ε) = { ε }

FOLLOW (S) = { **eof**, **c** }

FOLLOW (B) = { **c**, **eof** }

FOLLOW (D) = { **a**, **c** }

	a	**b**	**c**	**d**	**eof**
S	B **c** D B		B **c** D B	D B	
B	**a b**		**c** *S*		
D	ε		ε	**d**	

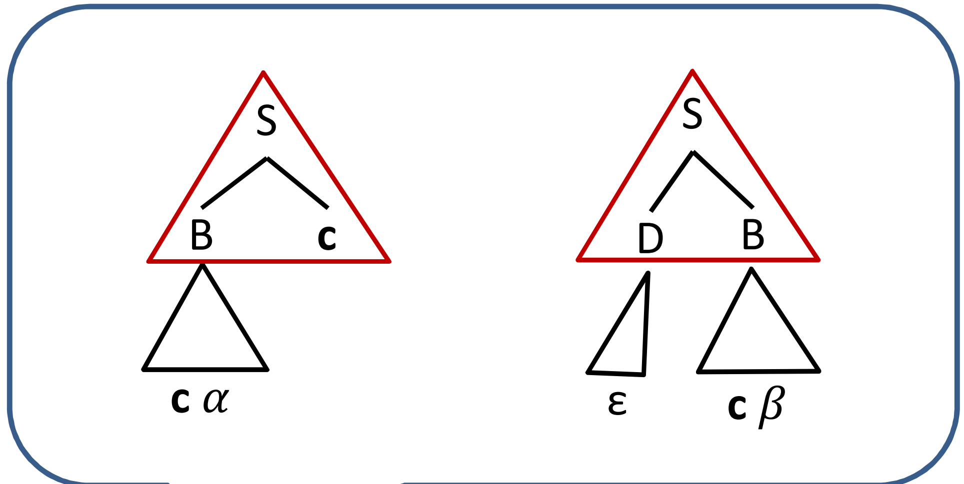
Why is a Table Collision a Problem?

<u>CFG</u>

S ⟶ **B c** | D B

B ⟶ **a b** | **c** S

D ⟶ **d** | ε



	a	**b**	**c**	**d**	e
S	B **c** D B		B **c** D B	D B	
B	**a b**		**c** *S*		
D	ε		ε	**d**