CS536

Building a Predictive Parser

Last Time: Intro LL(1) Predictive Parser

- “predict” the parse tree top-down
- Parser structure
  - 1 token of lookahead
  - A stack tracking parse tree 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 selector 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 LL(1) Grammar Transformations

- Necessary (but not sufficient conditions) for LL(1) Parsing:
  - Left factored
    - No rules with common prefix
    - Why? We'd need to look past the prefix to pick rule
  - Free of left recursion
    - No nonterminal loops for a production
    - Why? Need to look past list to know when to cap it

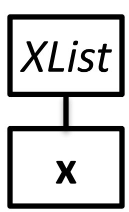
Why Left Recursion is a Problem (Blackbox View)

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

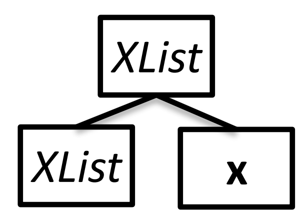
Current parse tree: $\boxed{XList}$

Current token: **x**

How should we grow the tree top-down?



(OR)



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

Correct if there are more **x**s

We don’t know which without more lookahead

Why Left Recursion is a Problem (Whitebox View)

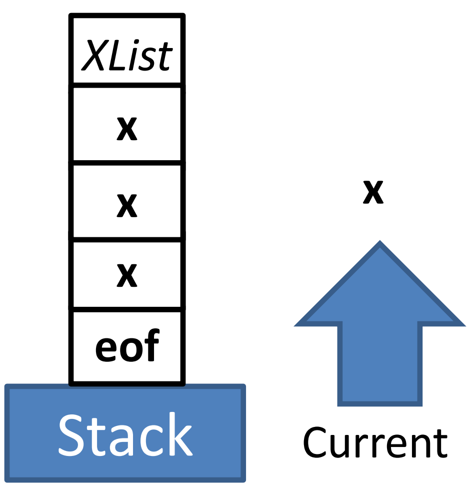
CFG snippet: *XList* ⟶ *XList* **x** | **x**

Current parse tree: $\boxed{XList}$

Current token: **x**

Parse table:

	x	**eof**
XList	*XList* **x**	**ε**



(Stack overflow)

Left Recursion Elimination: Review

Removing common prefix from grammar

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

With $A \longrightarrow \beta\ A'$

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

Where β does not start with *A* and may not be present

Preserve order (a list of α starting with β) but use right recursion

Left Recursion Elimination: Ex1

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

$E \longrightarrow E$ **cross id** | **id**

Left Recursion Elimination: Ex2

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

$$E \longrightarrow E + T \mid T$$
$$T \longrightarrow \mathrm{T} * \mathrm{F} \mid \mathrm{F}$$
$$F \longrightarrow (\ \mathrm{E}\ ) \mid \mathrm{id}$$

Left Recursion Elimination: Ex3

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

$SList \longrightarrow SList\ D \mid \varepsilon$

$D \longrightarrow Type$ **id semi**

$Type \longrightarrow$ **bool** | **int**

Left Factoring: Review

Removing common prefix from grammar

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

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

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

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

y_i May not be present, one of the β may be ε

Squash all "problem" rules starting with α together into one rule α *A'*

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

Left Factoring: Example 1

$$A \longrightarrow \alpha\ \beta_1 \mid \dots \mid \alpha\ \beta_m \mid y_1 \mid \dots \mid y_n \quad \Rightarrow \quad \begin{array}{l} A \longrightarrow \alpha\ A' \mid y_1 \mid \dots \mid y_n \\ A' \longrightarrow \beta_1 \mid \dots \mid \beta_m \end{array}$$

$$X \longrightarrow \mathbf{<a>} \mid \mathbf{<b>} \mid \mathbf{<c>} \mid \mathbf{d}$$

Left Factoring: Example 2

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

Stmt ⟶ **id assign** *E* | **id (** *EList* **)** | **return**

E ⟶ **intlit** | **id**

Elist ⟶ *E* | *E* **comma** *EList*

Left Factoring: Example 3

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

S ⟶ **if** *E* **then** *S* | **if** E **then** *S* **else** *S* | **semi**

E ⟶ **boollit**

Left Factoring: Not Always Immediate

$$A \longrightarrow \alpha\,\beta_1 \mid \ldots \mid \alpha\,\beta_m \mid y_1 \mid \ldots \mid y_n \quad \Rightarrow \quad \begin{array}{l} A \longrightarrow \alpha\,A' \mid y_1 \mid \ldots \mid y_n \\ A' \longrightarrow \beta_1 \mid \ldots \mid \beta_m \end{array}$$

This snippet yearns for left-factoring

$$\begin{array}{l} \mathrm{S} \longrightarrow \mathrm{A} \mid \mathrm{C} \mid \textbf{return} \\ \mathrm{A} \longrightarrow \textbf{id assign } E \\ \mathrm{C} \longrightarrow \textbf{id (} \mathit{EList} \textbf{)} \end{array}$$

but we cannot! At least without *inlining*

$$S \longrightarrow \textbf{id assign } E \mid \textbf{id (} \mathit{Elist} \textbf{)} \mid \textbf{return}$$

Let’s be more constructive

- So far, we’ve only talked about what <u>precludes</u> us from building a predictive parser
- It’s time to actually build the parse table

Building the Parse Table

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

(assume α is an arbitrary symbol string)

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

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, FIRST(α) =

$$\left\{ \mathbf{t} \,\middle|\, \left(\mathbf{t} \in \Sigma \wedge \alpha \overset{*}{\Rightarrow} \mathbf{t}\beta\right) \vee \left(\mathbf{t} = \varepsilon \wedge \alpha \overset{*}{\Rightarrow} \varepsilon\right) \right\}$$

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), it tells us that Production 1 is a viable choice
  - If **a** is in FIRST(β), it tells us that Production 2 is a viable choice
  - If **a** is in only FIRST(α) xor FIRST(β) , we can predict the rule we need.

FIRST Construction: Single Symbol

- We begin by doing FIRST sets for a <u>single</u>, 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 <u>also</u> 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)

FIRST(X) Example

Building FIRST(X) for nonterm X

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

Exp	⟶	*Term Exp'*
Exp'	⟶	**minus** *Term Exp'* | ε
Term	⟶	*Factor Term'*
Term'	⟶	**divide** *Factor Term'* | ε
Factor	⟶	**intlit** | **lparens** *Exp* **rparens**

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 $\text{FIRST}(Y_1) - \{\varepsilon\}$ in FIRST(α)
  - If ε is in $\text{FIRST}(Y_1)$: add $\text{FIRST}(Y_2) - \{\varepsilon\}$ to FIRST(α)
  - If ε is in $\text{FIRST}(Y_2)$: add $\text{FIRST}(Y_3) - \{\varepsilon\}$ 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 $\text{FIRST}(Y_1) - \{\varepsilon\}$
- If ε is in $\text{FIRST}(Y_{1 \text{ to } i-1})$: add $\text{FIRST}(Y_i) - \{\varepsilon\}$
- If ε is in all RHS symbols, add ε

$E \rightarrow TX$

$X \rightarrow +TX \mid \varepsilon$

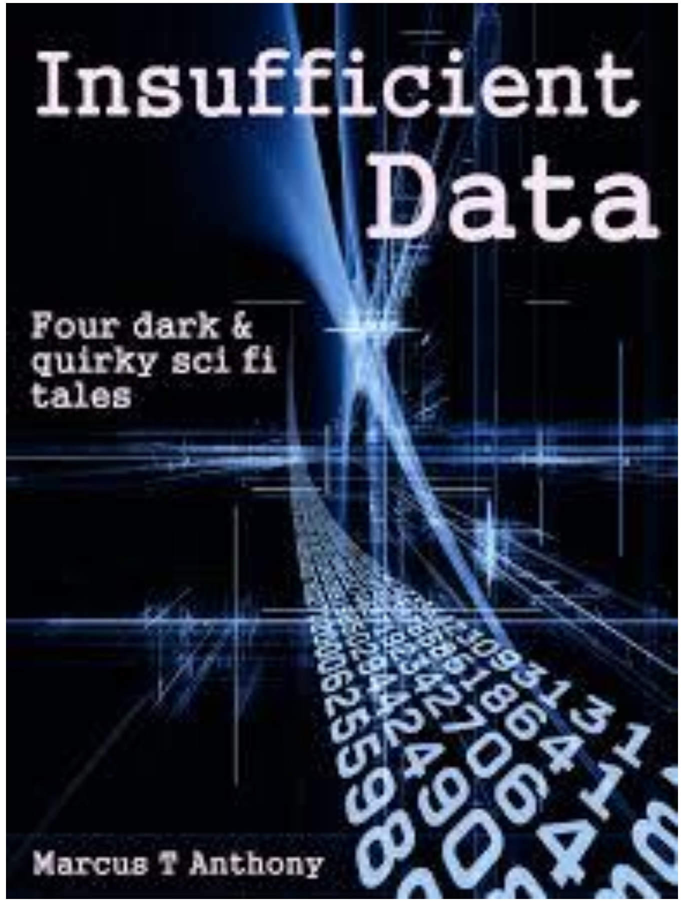
$T \rightarrow FY$

$Y \rightarrow *FY \mid \varepsilon$

$F \rightarrow (E) \mid id$

FIRST Sets aren't enough for Parse Tables

- If a rule can derive ε, we need to know what comes next
  - Obviously, some productions won't work



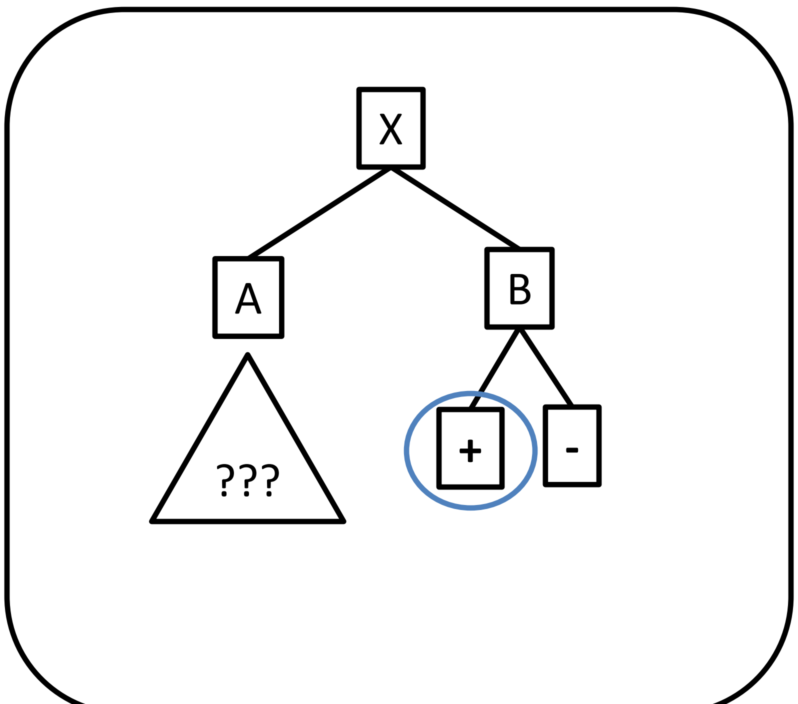
FOLLOW Sets

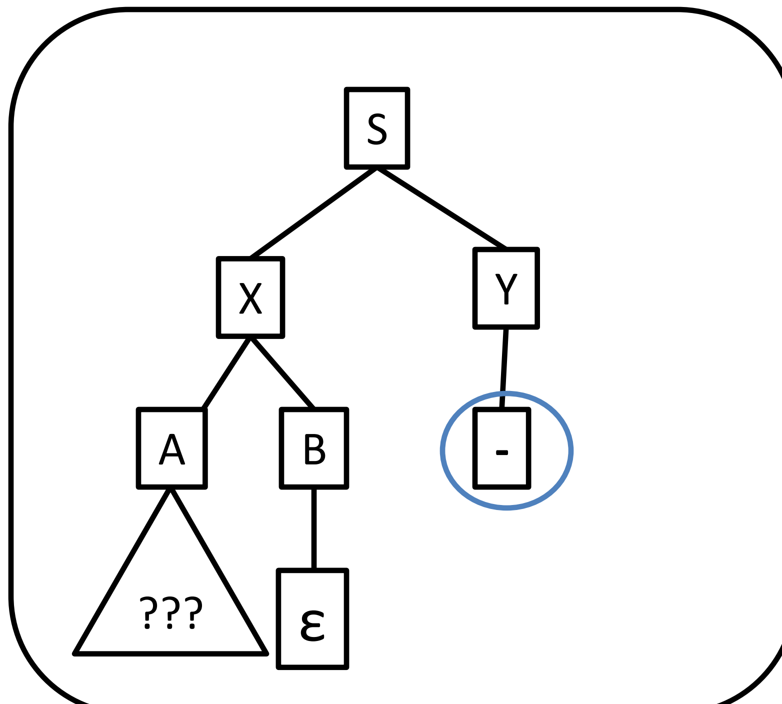
- For nonterminal A, FOLLOW(A) is the set of terminals that can appear immediately to the right of A
- Formally, FOLLOW(A) =

$$\left\{t \;\middle|\; \left(t \in \Sigma \wedge S \overset{+}{\Rightarrow} \alpha A t \beta\right) \vee \left(t = \mathbf{eof} \wedge S \overset{+}{\Rightarrow} \alpha A\right)\right\}$$

FOLLOW Sets: Pictorially

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

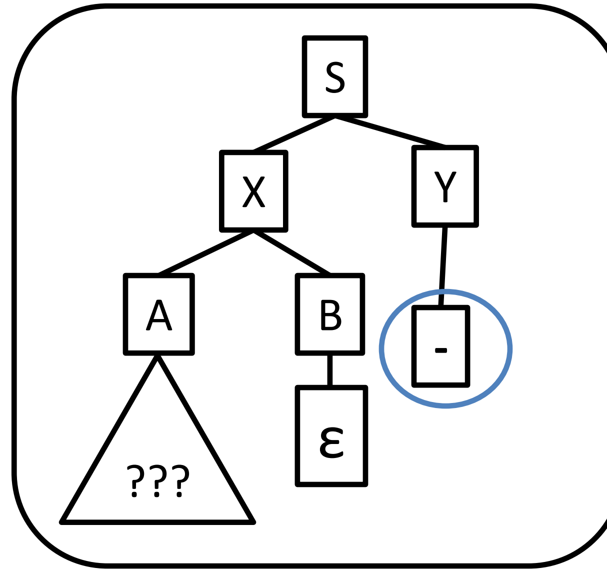


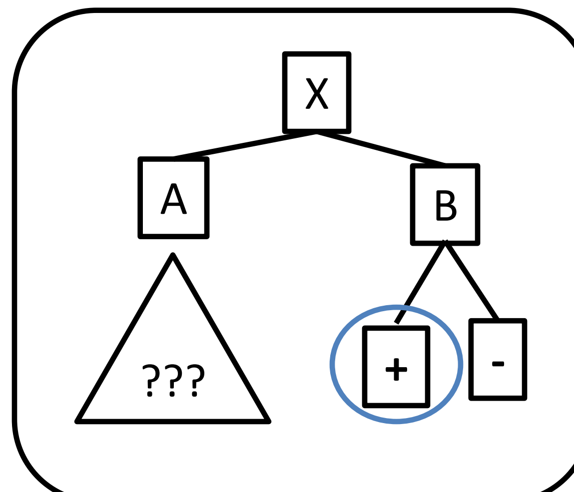


FOLLOW Sets: Construction

- 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)
- Continue building FOLLOW sets until saturation

Where α, β may be empty





FOLLOW Sets Example

<u>FOLLOW(A) for $X \longrightarrow \alpha\ A\ \beta$</u>
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

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 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

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

FIRST(α) for α = $Y_1 Y_2 \ldots Y_k$

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

$$S \rightarrow \mathbf{(}\ S\ \mathbf{)}\ |\ \mathbf{\{}\ S\ \mathbf{\}}\ |\ \boldsymbol{\varepsilon}$$

	(	**)**	**{**	**}**	**eof**
S					

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

$$S \rightarrow + S \mid \varepsilon$$

	+	eof
S		

How’s that Compiler Looking?