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

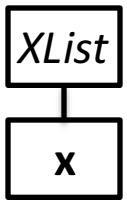
Why Left Recursion is a Problem (Blackbox View)

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

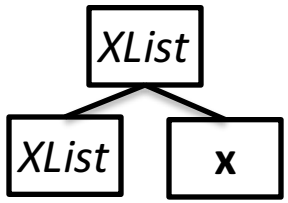
Current parse tree: *XList*

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)

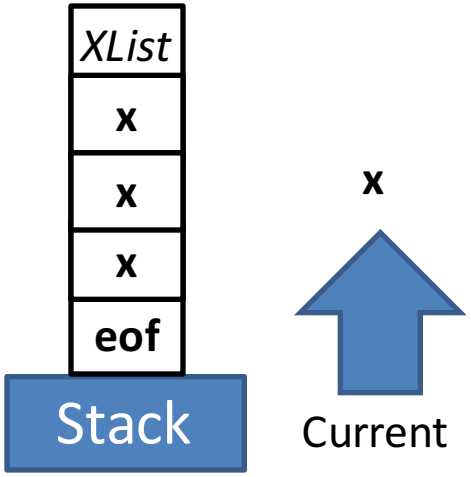
CFG snippet: $XList \longrightarrow XList\ \mathbf{x} \mid \mathbf{x}$

Current parse tree: *XList*

Current token: **x**

	x	**eof**
XList	*XList* **x**	ε

Parse table:



(Stack overflow)

Left Recursion Elimination: Review

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

With $A \longrightarrow \beta\ A'$

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

Head of the list

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

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

$E \longrightarrow E$ **cross id** $\mid$ **id**

α: **cross id** β: **id**

$$\begin{array}{l} E \longrightarrow \textbf{id}\,E' \\ E' \longrightarrow \textbf{cross id}\;E' \mid \varepsilon \end{array}$$

β: **id** α: **cross id**

Left Recursion Elimination: Ex2

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

$$\begin{array}{l} E \longrightarrow E + T \mid T \\ T \longrightarrow T * F \mid F \\ F \longrightarrow (\ \mathrm{E}\ ) \mid \mathbf{id} \end{array} \quad \Rightarrow \quad \begin{array}{l} E \longrightarrow TE' \\ E' \longrightarrow +\,\mathrm{TE}' \mid \varepsilon \\ T \longrightarrow FT' \\ T' \longrightarrow *\mathrm{FT}' \mid \varepsilon \\ F \longrightarrow (\ \mathrm{E}\ ) \mid \mathbf{id} \end{array}$$

Left Recursion Elimination: Ex3

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

SList ⟶ *SList D* | ε

D ⟶ *Type* **id semi**

Type ⟶ **bool** | **int**

⇒

SList ⟶ ε *SList'*

SList' ⟶ *D SList'* | ε

D ⟶ *Type* **id semi**

Type ⟶ **bool** | **int**

⇒

SList ⟶ *D SList* | ε

D ⟶ *Type* **id semi**

Type ⟶ **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 $\alpha A'$

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

Left Factoring: Example 1

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

$A \longrightarrow \alpha\,A' \mid \gamma_1 \mid \ldots \mid \gamma_n$

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

$\alpha \quad \beta_1 \quad \alpha \quad \beta_2 \quad \alpha \quad \beta_3 \quad \gamma_1$

$X \longrightarrow$ **< a > | < b > | < c > | d**

$\alpha \quad \gamma_1$

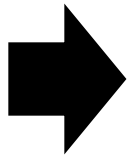
$X \longrightarrow$ **<** X' **| d**

$X' \longrightarrow$ **a > | b > | c >**

$\beta_1 \quad \beta_2 \quad \beta_3$

Left Factoring: Example 2

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



$A \longrightarrow \alpha\,A' \mid \gamma_1 \mid \ldots \mid \gamma_n$

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

β_1 β_2

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

E ⟶ **intlit** | **id**

Elist ⟶ *E* | *E* **comma** *EList*

Stmt ⟶ **id** *Stmt'* | **return**

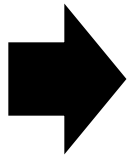
Stmt' ⟶ **assign** *E* | **(** *EList* **)**

E ⟶ **intlit** | **id**

Elist ⟶ *E* | *E* **comma** *EList*

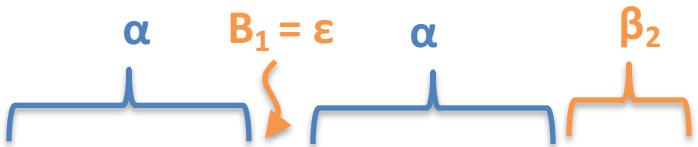
Left Factoring: Example 3

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



$A \longrightarrow \alpha\,A' \mid \gamma_1 \mid \ldots \mid \gamma_n$

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



$S \longrightarrow$ **if** E **then** S | **if** E **then** S **else** S | **semi**

$E \longrightarrow$ **boollit**

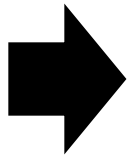
S $\longrightarrow$ **if** E **then** S S' | **semi**

S' $\longrightarrow$ **else** S | ε

E $\longrightarrow$ **boollit**

Left Factoring: Not Always Immediate

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



$A \longrightarrow \alpha\,A' \mid \gamma_1 \mid \ldots \mid \gamma_n$

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

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’ve only talked about what precludes 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 let 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 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)

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($Y_{1\ to\ i-1}$): 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(*Factor*) = { **intlit, lparens** }

FIRST(*Term'*) = { **divide,** ε }

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

FIRST(*Exp'*) = { **minus,** ε}

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

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)

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

Building FIRST(α)

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

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

FIRST(α) Example

Building FIRST(α)

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

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

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

FIRST(*TX*) = {**(**, **id**}
FIRST(**+***TX*) = { **+** }
FIRST(*FY*) = { **(**, **id** }
FIRST (*****FY*) = { ***** }
FIRST(**(** *E* **)**) = { **(** }
FIRST(**id**) = { **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

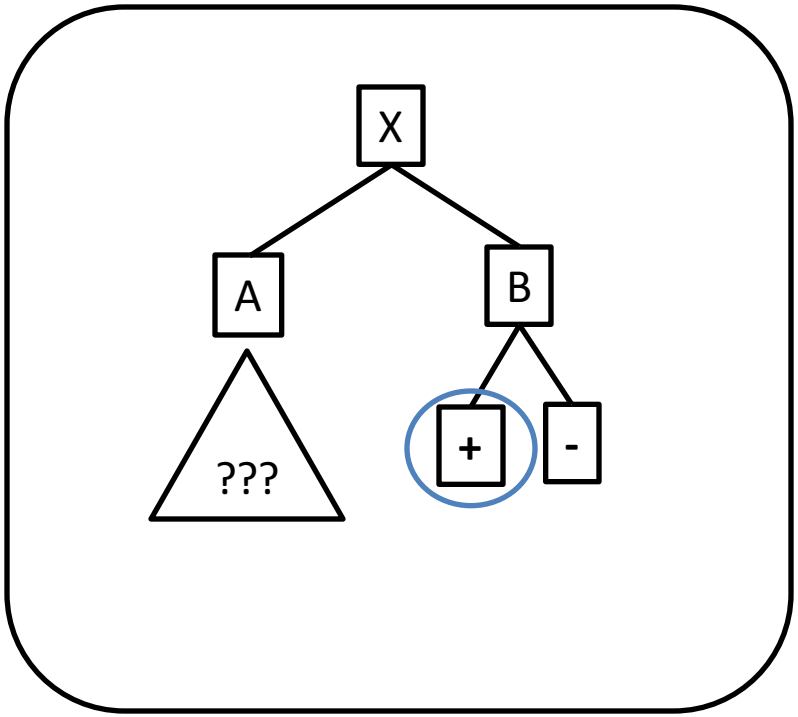
FOLLOW Sets

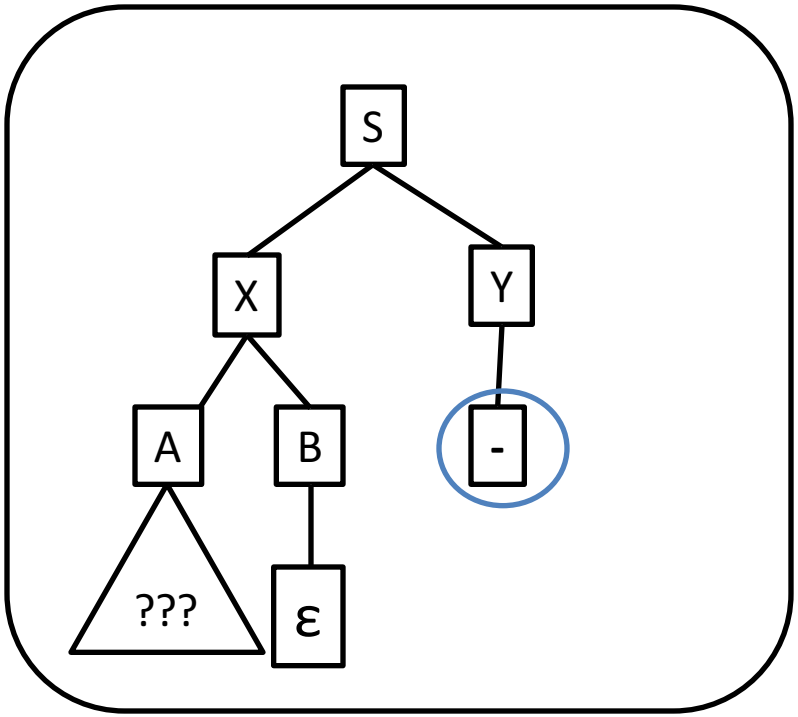
- For nonterminal A, FOLLOW(A) is the set of terminals that can appear immediately to the right of A
- Formally, FOLLOW(A) =

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

FOLLOW Sets: Pictorially

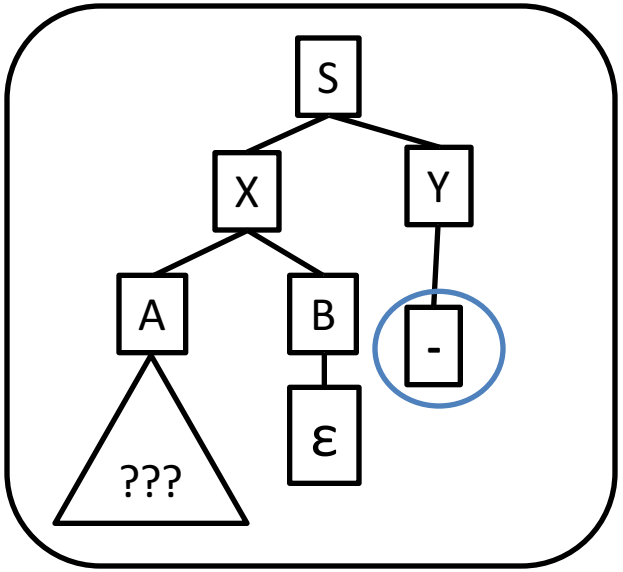
- For nonterminal A, FOLLOW(A) is the set of terminals that can appear immediately to the right of A

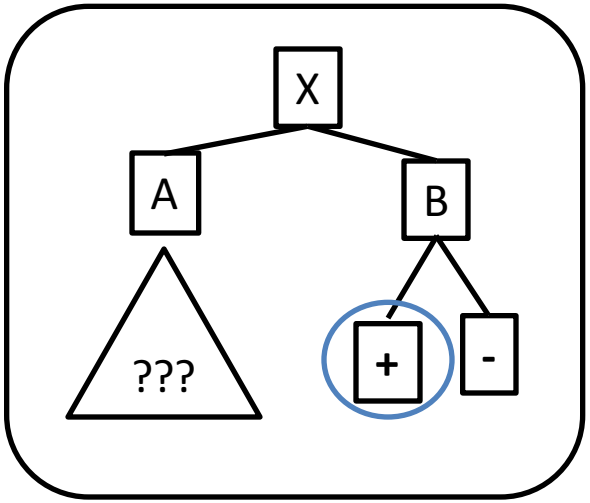




FOLLOW Sets: Construction

- To build FOLLOW(A)
  - If A is the start nonterminal, add **eof** **Where α, β may be empty**
  - 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





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** I D B
B ⟶ **a b** I **c** S
D ⟶ **d** I 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 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)

	a	**b**	**c**	**d**	**eof**
S	B **c** D B		B **c** D B	D B	
B	**a b**		**c** *S*		
D	ε		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**, **c** }
FOLLOW (B) = { **c**, **eof** }
FOLLOW (D) = { **a**, **c** }