Programming Language Syntax: Top-down Parsing

Read: Scott, Chapter 2.3.2 and 2.3.3

1

Lecture Outline

- *Top-down parsing, conclusion*
  - *Predictive parsing*
  - *LL(1) parsing table*
  - *FIRST, FOLLOW, and PREDICT sets*
  - *LL(1) grammars*

- *Quiz 2*

2

FIRST and FOLLOW sets

- Let α be any sequence of nonterminals and terminals
  - FIRST(α) is the set of terminals **a** that begin the strings derived from α. E.g., *expr* **$$** ⇒* **id**..., thus **id** in FIRST(*expr* **$$**)
  - If there is a derivation α ⇒* **ε**, then **ε** is in FIRST(α)
- Let *A* be a nonterminal
  - FOLLOW(*A*) is the set of terminals **b** (including special end-of-input marker **$$**) that can appear immediately to the right of *A* in some sentential form:

    start ⇒* …*A***b**… ⇒*…

⇒* IS ZERO OR MORE APPLICATIONS OF GRAMMAR PRODUCTIONS

Applying expr ⇒ term term_tail

expr $$ ⇒ term term_tail $$ ⇒ id factor_tail term_tail $$

Computing FIRST

Notation:
α is an arbitrary sequence of terminals and nonterminals

- Apply these rules until no more terminals or **ε** can be added to any FIRST(α) set

  (1) If α starts with a terminal **a**, then FIRST(α) = { **a** }

  (2) If α is a nonterminal *X*, where $X \rightarrow$ **ε**, then add **ε** to FIRST(α)

  (3) If α is a nonterminal $X \rightarrow Y_1Y_2\ldots Y_k$ then add **a** to FIRST(*X*) if for some *i*, **a** is in FIRST(Y_i) and **ε** is in all of FIRST(Y_1), … FIRST(Y_{i-1}). If **ε** is in all of FIRST(Y_1), … FIRST(Y_k), add **ε** to FIRST(*X*).

  ε

  - Everything in FIRST(Y_1) - **{ε}** is surely in FIRST(*X*)
  - If Y_1 does not derive **ε**, then we add nothing more; Otherwise, we add FIRST(Y_2) - **{ε}**, and so on

  Similarly, if α is $Y_1Y_2\ldots Y_k$, we'll repeat the above

Warm-up Exercise

start → *expr* **$$**
expr → *term term_tail*
term → **id** *factor_tail*
term_tail → **+** *term term_tail* | **ε**
factor_tail → ***** **id** *factor_tail* | **ε**

FIRST(*term*) = { **id** }

FIRST(*expr*) = {id}

FIRST(*start*) = {id}

FIRST(*term_tail*) = {+, ε}

FIRST(**+** *term term_tail*) = {+}

FIRST(*factor_tail*) = {*, ε}

Exercise

$\boxed{Ay \Rightarrow y \quad A \to \varepsilon}$ → E.g. we can derive string y from Ay

start → *S* **$$**
S → **x** *S* | *A* **y**
A → *BCD* | **ε**
B → **z** *S* | **ε**
C → **v** *S* | **ε**
D → **w** *S*

Ay => BCDy => CDy => Dy
=> w S y => w A y y =>
w y y

Compute FIRST sets:

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

FIRST(*A* **y**) = {y, z, v, w}

FIRST(*BCD*) = {z, v, w}

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

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

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

FIRST(*S*) = {x, y, z, v, w}

FIRST(*A*) = {ε, z, v, w}

FIRST(*B*) = {z, ε}

FIRST(*C*) = {v, ε}

FIRST(*D*) = {w}

Computing FOLLOW

Notation:
A,B,S are nonterminals.
α,β are arbitrary sequences of terminals and nonterminals.

- Apply these rules until nothing can be added to any FOLLOW(A) set
  - (1) If there is a production $A \to \alpha B\beta$, then everything in FIRST(β) except for ε should be added to FOLLOW(B)
  - (2) If there is a production $A \to \alpha B$, or a production $A \to \alpha B\beta$, where FIRST(β) contains ε, then everything in FOLLOW(A) should be added to FOLLOW(B)

Because: start $\Rightarrow^*$ … $A b$… $\Rightarrow$ … $\alpha B b$…

Thus $b \in$ FOLLOW(A) must be in FOLLOW(B) as well.

7

Warm-up

term_tail inherits FOLLOW(expr)

start $\to$ *expr* **$$**
expr $\to$ *term term_tail*
term $\to$ **id** *factor_tail*
term_tail $\to$ **+** *term term_tail* | ε
factor_tail $\to$ ***** **id** *factor_tail* | ε

FOLLOW(*expr*) = { **$$** }

FOLLOW(*term_tail*) = { $$ }

FOLLOW(*term*) = { +, $$ }

FIRST(term_tail) − {ε} ⊆ FOLLOW(term)
FOLLOW(expr) ⊆ FOLLOW(term)

FOLLOW(*factor_tail*) = { $$, + }

FOLLOW(term) ⊆ FOLLOW(factor_tail)

expr $$ ⇒ term term_tail $$ ⇒ term + term term_tail $$ ⇒ term + term $$

\+ in FOLLOW(term)

$$ in FOLLOW(term)

8

Exercise

Follow(S) ⊆ Follow(S)

start → S $$
S → **x** S | A **y**
A → BCD | **ε**
B → **z** S | **ε**
C → **v** S | **ε**
D → **w** S

Compute FOLLOW sets:

FOLLOW(*A*) = {y}

FOLLOW(*B*) = {v, w}

FOLLOW(*C*) = {w}

FOLLOW(*D*) = {y}

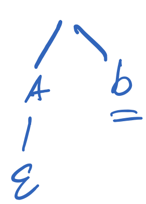
FOLLOW(*S*) = {$$, v, w, y}

PREDICT Sets

PREDICT(*A* → α) =

- **FIRST(α)** if α does not derive **ε**
- (FIRST(α) – {**ε**}) U FOLLOW(*A*) if α derives **ε**

{a, b, c}



Constructing LL(1) Parsing Table

- Algorithm uses PREDICT sets:

foreach production $A \rightarrow \alpha$ in grammar G
  foreach terminal **a** in PREDICT($A \rightarrow \alpha$)
    add $A \rightarrow \alpha$ into entry parse_table[A,**a**]

- If each entry in parse_table contains at most one production, then G is said to be LL(1)

Not LL(1) →

	a
A	$A \rightarrow \alpha$ $A \rightarrow \beta$

	a
A	$A \rightarrow \alpha$

Exercise

start $\rightarrow S$ **$$**	$B \rightarrow$ **z** S | **ε**
$S \rightarrow$ **x** S | A **y**	$C \rightarrow$ **v** S | **ε**
$A \rightarrow BCD$ | **ε**	$D \rightarrow$ **w** S

	x	y	z	v	w
S	xS	Ay	Ay	Ay	Ay

Compute PREDICT sets:

PREDICT($S \rightarrow$ **x** S) = $\{x\}$

PREDICT($S \rightarrow A$ **y**) = $\{y, z, v, w\}$ = FIRST(Ay)

PREDICT($A \rightarrow BCD$) = $\{z, v, w\}$

PREDICT($A \rightarrow$ **ε**) = $\{y\}$

… *etc*…

	x	y	z	v	w
A	—	ε	BCD	BCD	BCD

Writing an LL(1) Grammar

- Most context-free grammars are not LL(1) grammars
- Obstacles to LL(1)-ness
  - Left recursion is an obstacle. Why?
  - Common prefixes are an obstacle. Why?

expr → *expr* **+** *term* | *term*
term → *term* ***** **id** | **id**

stmt → **if b then** *stmt* **else** *stmt* |
if b then *stmt* |
a // ASSIGN

Removal of Left Recursion

- Left recursion can be removed from a grammar mechanically
- Started from this left recursive expression grammar:

expr → *expr* **+** *term* | *term*
term → *term* ***** **id** | **id**

- After removal of left recursion, we obtain this equivalent grammar, which is LL(1):

expr → *term term_tail*
term_tail → **+** *term term_tail* | ε
term → **id** *factor_tail*
factor_tail → ***** **id** *factor_tail* | ε

Removal of Common Prefixes

- Common prefixes can be removed mechanically as well by using left-factoring
- Original if-then-else grammar:

stmt → **if b then** *stmt* $\overbrace{\textbf{else}\ stmt}^{\alpha_1}$ |
if b then *stmt* $\underbrace{\ }_{\alpha_2}$ |
a

- After left-factoring:

stmt → **if b then** *stmt else_part* | **a**
else_part → **else** *stmt* | **ε**

Exercise

start → *stmt* **$$**
stmt → **if b then** *stmt else_part* | **a**
else_part → **else** *stmt* | **ε**

- Compute FIRSTs:

FIRST(*stmt* **$$**), FIRST(**if b then** *stmt else_part*), FIRST(**a**), FIRST(**else** *stmt*)

- Compute FOLLOW:

FOLLOW(*else_part*)

- Compute PREDICT sets for all 5 productions and fill in the LL(1) parsing table. Is the grammar LL(1)?

Exercise

start → *stmt* **$$**
stmt → **if b then** *stmt else_part* | **a**
else_part → **else** *stmt* | **ε**

- Compute FIRSTs:

FIRST(*stmt* **$$**) = { if, a }

FIRST(**if b then** *stmt else_part*) = { if }

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

FIRST(**else** *stmt*) = { else }

17

Exercise

start → *stmt* **$$**
stmt → **if b then** *stmt else_part* | **a**
else_part → **else** *stmt* | **ε**

- Compute FOLLOW:

PREDICT (else_part → else stmt) = { else }
PREDICT (else_part → ε) = { else, $$ }

FOLLOW(*else_part*) = { $$, else }

FOLLOW(else_part) = FOLLOW(stmt)

	...	else
else_part		else stmt ε

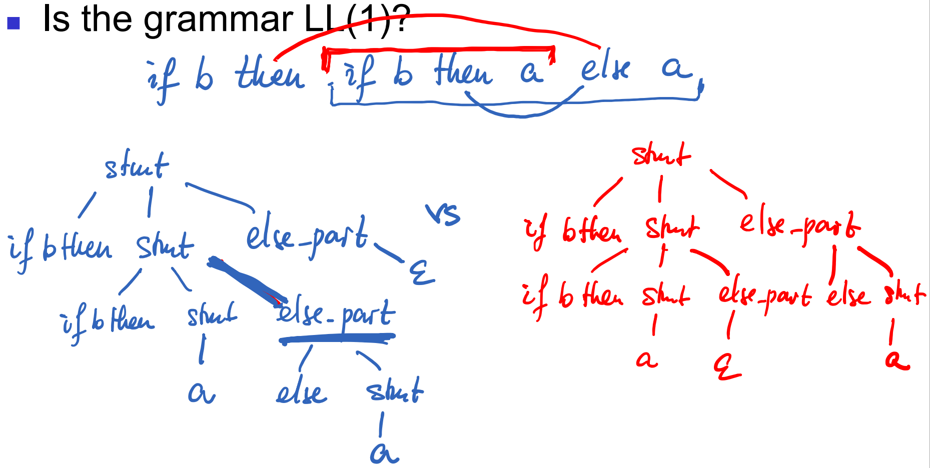
CONFLICT!

18

Exercise

start → *stmt* **$$**
stmt → **if b then** *stmt else_part* **|** **a**
else_part → **else** *stmt* **|** **ε**

- Is the grammar LL(1)?



Exercise