Constructing a Predict Table

Helper Function: Follow( )

- To understand Follow( ), we need to add a rule to our original grammar where a non-terminal derives ε, e.g. rule 7: $B \to \varepsilon$
- Now we can derive:
  $S' \overset{1}{\Rightarrow} \vdash S \dashv \overset{2}{\Rightarrow} \vdash AyB\dashv \overset{3}{\Rightarrow} \vdash abyB\dashv \overset{7}{\Rightarrow} \vdash aby\dashv$
- *key point:* ⊣ can appear after the B *but there is no derivation* $B \Rightarrow^* \dashv$
- i.e. *using First( ) is not sufficient*
  - the symbol '⊣' came from rule 1: $S' \to \vdash S \dashv$
  - the symbol B came from rule 2: $S \to AyB$
  - and B derives ε with rule 7: $B \to \varepsilon$
- *conclusion:* ⊣ is in the follow set of B

1. $S' \to \vdash S \dashv$
2. $S \to AyB$
3. $A \to ab$
4. $A \to cd$
5. $B \to z$
6. $B \to wz$
7. $B \to \varepsilon$

Constructing a Predict Table

Using Follow( ) to Construct the Predict Table

- The Predict Table for our new grammar has a new entry Predict(B, ⊣) = 7 (the rest is the same)

1. $S' \rightarrow \vdash S \dashv$
2. $S \rightarrow AyB$
3. $A \rightarrow ab$
4. $A \rightarrow cd$
5. $B \rightarrow z$
6. $B \rightarrow wz$
7. $B \rightarrow \varepsilon$

	a	b	c	d	y	w	z	⊢	⊣
S’								1	
S	2		2						
A	3		4						
B						6	5		7

- We used rule 7 to take the step $\vdash abyB \dashv \;\Rightarrow\; \vdash aby \dashv$
- So if B is on the stack and the next input symbol is ‘⊣’ then expand with rule 7, i.e. have B derive the empty string.

Constructing a Predict Table

Helper Function: Follow( )

- The terminal symbol '⊣' is in Follow(B) because there is a derivation from the start symbol $S' \Rightarrow^* \vdash abyB\dashv$
- *Informally:* Follow(N) is the set of terminals c that can follow N in some derivation; that is, $S \Rightarrow^* \cdots Nc \cdots$
- *Formally:* for any non-terminal N, Follow(N) = $\{ c \mid S' \Rightarrow^* \alpha N c \beta \}$
  - where α and β are (possibly empty) sequences of terminals and non-terminals
- But Follow(N) is only relevant if there is a derivation $N \Rightarrow^* \varepsilon$ so we need to check if N can derive the empty string.
- We need yet another helper function Nullable( ), sometimes called Empty( )…

Constructing a Predict Table

Helper Function: Nullable( )

- *Informally*: Nullable(N) indicates that N can derive the empty string, i.e. $N \Rightarrow^* \varepsilon$
- *More generally, ask* if α can derive the empty string where α is in (terminals | non-terminals)* and each B_i below is a single terminal or non-terminal.
- *Formally*: Nullable(α) = true if $\alpha \Rightarrow^* \varepsilon$
  - False if α has a terminal in it (only non-terminals can derive ε)
  - True if there is a rule $\alpha \rightarrow \varepsilon$
  - For any rule of the form $\alpha \rightarrow B_1 B_2 \cdots B_n$
    Nullable(α) is true if each of Nullable(B_1), Nullable(B_2), ..., Nullable(B_n) is true.

LL(1) Parsing

```
Input: w
push S' (start symbol) on stack
for each a ∈ w {
    while (top of stack is a non-terminal N ) {   // 1st try expand
        if ( Predict(N, a) == (N → α) )
            pop N
            push α on stack (in reverse)
        else
            reject                                 //  no rule found
    }
    c = pop_stack()                                //  2nd try match
    if (c ≠ a)
        reject                                     //  no match found
}
accept w
```

Example of LL(1) Parsing

LL(1) Parsing: Parse ⊢ cdy ⊣

	Derivation	Read	Input	Stack	Action
1	S’		⊢ cdy ⊣	> **S’**	predict(S’, ⊢) = 1
2	⊢ S ⊣		**⊢** cdy ⊣	> **⊢** S ⊣	match
3	⊢ S ⊣	⊢	cdy ⊣	> **S** ⊣	predict(S, c) = 2
4	⊢ AyB ⊣	⊢	cdy ⊣	> **A** y B ⊣	predict(A, c) = 4
5	⊢ cdyB ⊣	⊢	**c**dy ⊣	> **c** d y B ⊣	match
6	⊢ cdyB ⊣	⊢ c	**d**y ⊣	> **d** y B ⊣	match
7	⊢ cdyB ⊣	⊢ cd	**y** ⊣	> **y** B ⊣	match
8	⊢ cdyB ⊣	⊢ cdy	⊣	> **B** ⊣	predict(B,⊣) = 7
9	⊢ cdy ⊣	⊢ cdy	⊣	> **⊣**	match
10	⊢ cdy ⊣	⊢ cdy ⊣		>	ACCEPT

Again ‘>’ indicates the top of the stack.

More about Follow( )

Helper Function: Follow( ) is Complicated

- Need a different grammar to see this fact.
- In the grammar on the right $\dashv$ ∈ Follow(S) since S' → ⊢ S ⊣.
- But ⊣ ∈ Follow(B) *even though there is no rule of the form* S' → ⋯ B ⊣ since we have S' ⇒ ⊢ S ⊣ ⇒ ⊢ ABC ⊣ ⇒ ⊢ AB ⊣
- I.e. in the rule S → ABC, since Nullable(C) = true and ⊣ ∈ Follow(S) therefore ⊣ ∈ Follow(B).

1. S' → ⊢ S ⊣
2. S → ABC
3. A → aA
4. A → ε
5. B → bB
6. B → ε
7. C → cC
8. C → ε

- More generally if N → $B_1 B_2 \ldots B_i B_{i+1} \ldots B_n$ and Nullable($B_{i+1} B_{i+2} \ldots B_n$) then Follow($B_i$) includes Follow(N) i.e. if the RHS of B_i *is nullable, then what follows N can also follow* B_i.

More about Follow( )

Helper Function: Follow( ) is Complicated

- *Asking:* Starting from the start symbol, does the terminal c ever occur immediately following B_i.
- Here c is a terminal; A, N are non-terminals; B_i is a single terminal or non-terminal; $\alpha, \beta \in$ (terminals | non-terminals)*
- **Follow**$(B_i) = \{ c \mid S \Rightarrow^* \alpha B_i c \beta \}$

  Initialize: Follow(N) = { } for all non-terminals N // *the empty set*

  for each rule of the form $A \rightarrow B_1 B_2 \ldots B_{i-1} B_i B_{i+1} \ldots B_k$:

      for i = 1 to k:

          if (B_i is a non-terminal) // *what can appear after* B_i*?*

              $\text{Follow}(B_i) = \text{Follow}(B_i) \cup \text{First}(B_{i+1} B_{i+2} \ldots B_k)$

              if ($\text{Nullable}(B_{i+1} B_{i+2} \ldots B_k)$) // *what can appear after* A*?*

                  $\text{Follow}(B_i) = \text{Follow}(B_i) \cup \text{Follow}(A)$

Constructing a Predict Table

Constructing Predict(N, c)

- *Key Task, asking* if N is on the top of the stack and c is the next symbol in the input, which rule should be used to expand N?
- Here α, β $\in$ (terminals | non-terminals)*

  c is a terminal, N is a non-terminal
- **Predict**(N, c) = { the rule N $\rightarrow$ α | c $\in$ First(α) } $\cup$

  { the rule N $\rightarrow$ β | c $\in$ Follow(N) and Nullable(β) = true }
- *In summary:* To fill out the Predict Table, i.e. calculate which rule to use for Predict(N, c), we need to consider
  - First(α) for all rules of the form N $\rightarrow$ α
  - Follow(N) for all rules of the form N $\rightarrow$ β whenever Nullable(β) is true.

Example of Constructing a Predict Table

First( )

First(α) ={ a | $\alpha \Rightarrow^*$ aβ}

A: a $\in$ First (A) since A $^3\Rightarrow$ aA

B: b $\in$ First (B) since B $^5\Rightarrow$ bB

S: a $\in$ First (S) since S $^2\Rightarrow$ AB $^3\Rightarrow$ aAB

b $\in$ First (S) since S $^2\Rightarrow$ AB $^4\Rightarrow$ B $^5\Rightarrow$ bB

Nullable( )

Nullable(α) = true if $\alpha \Rightarrow^* \varepsilon$

A: Nullable(A) = true since A $^4\Rightarrow \varepsilon$

B: Nullable(B) = true since B $^6\Rightarrow \varepsilon$

S: Nullable(S) = true since S $^2\Rightarrow$ AB $^4\Rightarrow$ B $^6\Rightarrow \varepsilon$

1. S' $\rightarrow$ ⊢ S ⊣
2. S $\rightarrow$ AB
3. A $\rightarrow$ aA
4. A $\rightarrow$ ε
5. B $\rightarrow$ bB
6. B $\rightarrow$ ε

Example of Constructing a Predict Table

Follow( )

Recall: Follow(B_i) = { c | S' $\Rightarrow^*$ $\alpha B_i c \beta$}

If Nullable(B_i) we need to consider Follow(B_i)

for rules N $\rightarrow$ $B_1 B_2 \ldots B_{i-1} B_i B_{i+1} \ldots B_n$:

(i) Follow(B_i) includes First ($B_{i+1} B_{i+2} \ldots B_n$)

(ii) if (Nullable($B_{i+1} B_{i+2} \ldots B_n$))
Follow(B_i) includes Follow(N)

1. S' $\rightarrow$ ⊢ S ⊣
2. S $\rightarrow$ AB
3. A $\rightarrow$ aA
4. A $\rightarrow$ ε
5. B $\rightarrow$ bB
6. B $\rightarrow$ ε

S: ⊣ $\in$ Follow(S) since S' $\rightarrow$ ⊢ S ⊣ and ⊣ $\in$ First(⊣) by (i)

B: ⊣ $\in$ Follow(B) since S $\rightarrow$ AB and ⊣ $\in$ Follow(S) by (ii)

A: ⊣ $\in$ Follow(A) since S $\rightarrow$ AB, Nullable(B) and ⊣ $\in$ Follow(S) by (ii)

b $\in$ Follow(A) since S $\rightarrow$ AB and b $\in$ First(B) by (i)

Example of Constructing a Predict Table

The Predict Table

- Let N $\in$ {S, A, B} and let c $\in$ {a, b, ⊢, ⊣}
- For the entries due to First(N), use rule N $\rightarrow \alpha$ where c $\in$ First(α)
- For the entries due to Follow(N) use rule N $\rightarrow \alpha$ where c $\in$ Follow(N) and Nullable(α) = true

Grammar

1. S' $\rightarrow$ ⊢ S ⊣
2. S $\rightarrow$ AB
3. A $\rightarrow$ aA
4. A $\rightarrow \varepsilon$
5. B $\rightarrow$ bB
6. B $\rightarrow \varepsilon$

Predict Table

	a	b	⊢	⊣
S'			1	
S	2	2		2
A	3	4		4
B		5		6

Computing Nullable

Nullable( )

```
for each non-terminal A: Nullable(A) = false        // initialize
repeat
  for each rule A → B_1 B_2 ... B_k                  // check rules
    if (k = 0) or (Nullable(B_1) = ··· = Nullable(B_k) = true)
      then Nullable(A) = true                       // k=0 means
until nothing changes                                // empty string
```

R1 $S' \to \vdash S \dashv$

R2 $S \to b\ S\ d$

R3 $S \to p\ S\ q$

R4 $S \to C$

R5 $C \to c\ C$

R6 $C \to \varepsilon$

Iteration	0	1	2	3
S'	false	false	false	false
S	false	false	true	true
C	false	true	true	true

Computing First

First(A) **for a Non-terminal** A

```
for each non-terminal A: First(A) = { }      // initialize
repeat
  for each rule A → B1B2···Bk                // check rules
    for i = 1 … k
      if (Bi is a non-terminal) then         // Bi is a non-terminal
        First(A) = First(A) ∪ First(Bi)
        if (not Nullable(Bi)) then break;    // go to next rule
      else                                   // Bi is a terminal
        First(A) = First(A) ∪ {Bi};
        break                                // go to next rule
until nothing changes
```

General Idea: keep processing $B_1B_2\cdots B_k$ until you encounter a terminal or a symbol that is not Nullable. Then go to the next rule.

Computing First

First*($B_1B_2\cdots B_k$) **for a Concatenation of Symbols**

// Before you considered each rule, now just consider $B_1B_2\cdots B_k$.

```
answer = { }                                   // initialize
for i = 1 … k                                  // check B_1B_2···B_k
  if (B_i is a non-terminal) then              // B_i is a non-terminal
    answer = answer ∪ First(B_i)
    if (not Nullable(B_i)) then break          //  return answer
  else                                         // B_i is a terminal
    answer = answer ∪ {B_i}
    break;                                     // return answer
return answer;
```

General Idea: keep processing $B_1B_2\cdots B_k$ until you encounter a terminal or a symbol that is not Nullable. Then go to the next rule.

Computing First

First(A) **for a Non-terminal** A

R1 S' → ⊢ S ⊣
R2 S → b S d
R3 S → p S q
R4 S → C
R5 C → c C
R6 C → ε

Iteration	**0**	**1**	**2**	**3**
S'	{ }	{⊢}	{⊢}	{⊢}
S	{ }	{b, p}	{b, c, p}	{b, c, p}
C	{ }	{c}	{c}	{c}

- Iteration 0: set all to empty set (line 1)
- Iteration 1: With rules R1, R2, R3, and R5 set the values using lines 8-9 with i=1.
- Iteration 2: c becomes part of First(S) using line 4 and R4 namely First(S) = First(S) ∪ First{C}
- Iteration 3: nothing changes so terminate

Computing Follow

Follow(A) **for a Non-terminal** A

for each non-terminal A except S': Follow(A) = { } // initialize
repeat
  for each rule $A \rightarrow B_1 B_2 \cdots B_k$ // check rules
    for i = 1 … k
      if (B_i is a non-terminal) // B_i is a non-terminal
        $\text{Follow}(B_i) = \text{Follow}(B_i) \cup \text{First}^*(B_{i+1} \cdots B_k)$ // case (i)
        if ($\text{Nullable}(B_{i+1} \cdots B_k)$) **then**
          $\text{Follow}(B_i) = \text{Follow}(B_i) \cup \text{Follow}(A)$ // case (ii)
until nothing changes

- No terminal can follow S', so no need to calculate its follow set.
- Have two cases for Follow(B_i): case (i) $\text{First}^*(B_{i+1} \cdots B_k)$
  case (ii) $\text{Nullable}(B_{i+1} \cdots B_k)$

Computing Follow

Follow(A) **for a Non-terminal** A

R1 $S' \rightarrow \vdash S \dashv$
R2 $S \rightarrow b\ S\ d$
R3 $S \rightarrow p\ S\ q$
R4 $S \rightarrow C$
R5 $C \rightarrow c\ C$
R6 $C \rightarrow \varepsilon$

Iteration	**0**	**1**	**2**
S	{ }	{⊣, d, q}	{⊣, d, q}
C	{ }	{⊣, d, q}	{⊣, d, q}

- Iteration 0: set all to empty set (line 1)
- Iteration 1: with R1, R2 and R3 set the values S (lines 3-6)
  with R4 Follow(C) = Follow(C) ∪ Follow{S} (line 8)
- Iteration 3: nothing changes so terminate

Example of Constructing a Predict Table

The Predict Table

- Let N $\in$ {S', S, C} and let c $\in$ {b, c, d, p, q, ⊢, ⊣}
- For the entries due to First(N), use rule N $\rightarrow \alpha$ where c $\in$ First(α) (blue entries in table).
- For the entries due to Follow(N) use rule N $\rightarrow \alpha$ where c $\in$ Follow(N) and Nullable(α) = true (black entries in table).

Grammar

R1 S' $\rightarrow$ ⊢ S ⊣
R2 S $\rightarrow$ b S d
R3 S $\rightarrow$ p S q
R4 S $\rightarrow$ C
R5 C $\rightarrow$ c C
R6 C $\rightarrow \varepsilon$

Predict Table

	b	c	d	p	q	⊢	⊣
S'						1	
S	2	4	4	3	4		4
C		5	6		6		6

LL(1) Summary

- **Goal:** Determine which rule to use when N (a non-terminal) is on the stack and c (a terminal) is the next symbol in the input. I.e. what is Predict(N, c)?
- **First():** Which terminals can begin a string derived from α?
  - First(α) = $\{ c \mid \alpha \Rightarrow^* c\beta \}$ where c is a terminal and $\alpha, \beta, \gamma \in$ (terminals | non-terminals)*.
- **Nullable():** Can α derive the empty string?
  - Nullable(α) = true if $\alpha \Rightarrow^* \varepsilon$.
- **Follow():** Which terminals that can follow N in some derivation?
  - Follow(N) = $\{ c \mid S' \Rightarrow^* \alpha N c \gamma \}$. I.e. for some rule $S \rightarrow \alpha N \beta$

    (i) $c \in$ First(β)

    (ii) Nullable(β) and $c \in$ Follow(S)

LL(1) Summary + Example

Predict(N, c) = { the rule $N \to \alpha$ | $c \in$ First(α) } ∪
{ the rule $N \to \beta$ | Nullable(β) and $c \in$ Follow(N) }

Examples of First, Nullable and Follow

1. $S' \to \vdash S \dashv$
2. $S \to AB$
3. $A \to aA$
4. $A \to \varepsilon$
5. $B \to bB$
6. $B \to \varepsilon$

$b \in$ First (B) since $B \overset{5}{\Rightarrow} bB$

Nullable(B) = true since $B \overset{6}{\Rightarrow} \varepsilon$

(i) $b \in$ Follow(A) since $S \to AB$ and $b \in$ First(B)

(ii) $\dashv \in$ Follow(A) since $S \to AB$, Nullable(B)
and $\dashv \in$ Follow(S)
I.e. we have $S' \Rightarrow \vdash S \dashv \Rightarrow \vdash AB \dashv \Rightarrow \vdash A \dashv$

Check your work at http://smlweb.cpsc.ucalgary.ca/start.html