

CS311—Computational Structures

# Top-down parsing with a Parse Table

Lecture 11





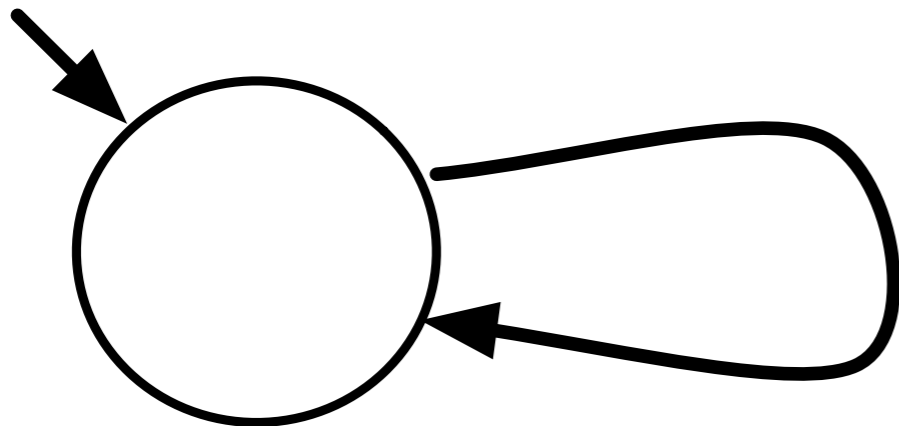




# Recall the theory:

$\frac{}{\text{push}(S)}$

where  $S$  is the grammar's start symbol



$\frac{\Lambda, A}{\text{pop, push}(\omega)}$  for each rule  $A \rightarrow \omega$ ,  $\omega$  a sequence of terminals and variables

$\frac{a, a}{\text{pop}}$  for each terminal  $a \in A$

- At each step, PDA can either
  1. read input  $a$ , iff  $a$  is on the stack (match), or
  2. replace  $A$  on the stack with  $\omega$ , where  $A \rightarrow \omega$  is a rule of the grammar (derive).
- How to choose *which*  $A \rightarrow \omega$ ?

# Recall the Practice

```
and moreConcat () =  
  if (couldBeSimple (!lookahead))  
  then  
    let val x = closure()  
        val y = moreConcat()  
    in case y of  
        NONE => SOME x  
      | SOME z => SOME(Concat(x,z))  
    end  
  else NONE  
  
and couldBeSimple LeftParen = true  
| couldBeSimple Hash = true  
| couldBeSimple Zero = true  
| couldBeSimple (Single _) = true  
| couldBeSimple _ = false
```

```
moreConcat →  
  closure moreConcat  
  | Λ
```

# Top-down predictive parsers

- Use an explicit stack instead of recursion.
- Represent the transitions of the PDA in a table, rather than as code
- Choice of action of the parser (which production to use to expend the stack symbol) represented by three functions:
  - Nullable
  - First
  - Follow

- **Nullable:** Can a symbol derive the empty string?  
False for every terminal symbol.
  - » Nullable: (term or nonterm)  $\rightarrow$  bool
- **First:** all the terminals that a non-terminal could possibly derive as its first symbol.
  - » First: (term or nonterm)  $\rightarrow$  set( term )
  - » First: sequence(term + nonterm)  $\rightarrow$  set( term )
- **Follow:** all the terminals that could immediately follow the string derived from a non-terminal.
  - » Follow: non-term  $\rightarrow$  set( term )

# Example First and Follow Sets

$E \rightarrow T E' \$$

$E' \rightarrow + T E'$

$E' \rightarrow \Lambda$

$T \rightarrow F T'$

$T' \rightarrow * F T'$

$T' \rightarrow \Lambda$

$F \rightarrow ( E )$

$F \rightarrow \text{id}$

First E = { "(", "id" }

First F = { "(", "id" }

First T = { "(", "id" }

First E' = { "+",  $\Lambda$  }

First T' = { "\*",  $\Lambda$  }

Follow E = { ")", "\$" }

Follow F = { "+", "\*", ")", "\$" }

Follow T = { "+", ")", "\$" }

Follow E' = { ")", "\$" }

Follow T' = { "+", ")", "\$" }

- First of a terminal is itself.
- First can be extended to be defined on a *sequence* of symbols.

# Nullable

- if  $\Lambda$  is in  $\text{First}(\text{symbol})$  then that symbol is *nullable*.
- Sometimes, rather than let  $\Lambda$  be a symbol, we use an additional function Nullable.
  - ▶  $\text{Nullable}(E') = \text{true}$
  - ▶  $\text{Nullable}(T') = \text{true}$
  - ▶ Nullable for all other symbols is false

$E$	$\rightarrow$	$T E' \$$
$E'$	$\rightarrow$	$+ T E'$
$E'$	$\rightarrow$	$\Lambda$
$T$	$\rightarrow$	$F T'$
$T'$	$\rightarrow$	$* F T'$
$T'$	$\rightarrow$	$\Lambda$
$F$	$\rightarrow$	$( E )$
$F$	$\rightarrow$	$\text{id}$

# Computing *FIRST*

- Use the following rules until no more terminals can be added to any *FIRST* set.
  - 1) if  $X$  is a terminal,  $FIRST(X) = \{X\}$
  - 2) if  $X \rightarrow \Lambda$  is a production then add  $\Lambda$  to  $FIRST(X)$ ,  
(or: set nullable of  $X$  to true).
  - 3) if  $X$  is a non-terminal and  $X \rightarrow Y_1 Y_2 \dots Y_k$ 
    - add  $a$  to  $FIRST(X)$ 
      - » if  $a$  in  $FIRST(Y_i)$  and
      - » for all  $j < i$ . nullable ( $Y_j$ )
        - equivalently,  $\Lambda$  in  $FIRST(Y_j)$
    - e.g., if  $Y_1$  can derive  $\Lambda$  then,  
if  $a$  is in  $FIRST(Y_2)$ ,  $a$  is surely in  $FIRST(X)$  as well.

# Example First Computation

- Terminals

- $\text{First}(\$) = \{\$\}$   $\text{First}(\ast) = \{\ast\}$   $\text{First}(+) = \{+\}$  ...

- Empty Productions

- add  $\Lambda$  to  $\text{First}(E')$ , add  $\Lambda$  to  $\text{First}(T')$

- Other NonTerminals

- Computing from the lowest layer (F) up:

- »  $\text{First}(F) = \{\text{id}, (\}$

- »  $\text{First}(T') = \{\Lambda, \ast\}$

- »  $\text{First}(T) = \text{First}(F) = \{\text{id}, (\}$

- »  $\text{First}(E') = \{\Lambda, +\}$

- »  $\text{First}(E) = \text{First}(T) = \{\text{id}, (\}$

$E$	$\rightarrow$	$T E' \$$
$E'$	$\rightarrow$	$+ T E'$
$E'$	$\rightarrow$	$\Lambda$
$T$	$\rightarrow$	$F T'$
$T'$	$\rightarrow$	$\ast F T'$
$T'$	$\rightarrow$	$\Lambda$
$F$	$\rightarrow$	$( E )$
$F$	$\rightarrow$	$\text{id}$

# Computing FOLLOW

- Use the following rules until nothing can be added to any follow set:

1) Place \$ (the end of input marker) in FOLLOW(S) where S is the start symbol.

*(This is Hein's rule 1)*

2) If  $A \rightarrow a B b$ , then put everything in FIRST( $b$ ) except  $\Lambda$  into FOLLOW(B)

*(This is Hein's rule 3)*

3) If there is a production  $A \rightarrow a B$ , or  $A \rightarrow a B b$  where FIRST( $b$ ) contains  $\Lambda$  (i.e., nullable  $b$ ), then put everything in FOLLOW(A) into FOLLOW(B)

*(This is a combination of Hein's rules 2 & 4)*

# Example: Follow Computation

- Rule 1, Start symbol
  - Add \$ to Follow(E)
- Rule 2, Productions with embedded non-terminals
  - Add First( ) = { ) } to Follow(E)
  - Add First(\$ ) = { \$ } to Follow(E')
  - Add First(E') = { +,  $\Lambda$  } to Follow(T)
  - Add First(T') = { \*,  $\Lambda$  } to Follow(F)
- Rule 3, Non-terminal in rightmost position
  - Add follow(E') to follow(E') (doesn't do much)
  - Add follow (T) to follow(T')
  - Add follow(T) to follow(F) since  $T' \Rightarrow \Lambda$
  - Add follow(T') to follow(F) since  $T' \Rightarrow \Lambda$

$E$	$\rightarrow$	$T E' \$$
$E'$	$\rightarrow$	$+ T E'$
$E'$	$\rightarrow$	$\Lambda$
$T$	$\rightarrow$	$F T'$
$T'$	$\rightarrow$	$* F T'$
$T'$	$\rightarrow$	$\Lambda$
$F$	$\rightarrow$	$( E )$
$F$	$\rightarrow$	$id$

# Table from First and Follow

For each production  $A \rightarrow \omega$  do steps 1–3 below:

1. For each terminal  $a$  in  $\text{First } \omega$  add  $A \rightarrow \omega$  to  $M[A,a]$
2. If  $\Lambda$  is in  $\text{First } \omega$ , add  $A \rightarrow \omega$  to  $M[A,b]$  for each terminal  $b$  in  $\text{Follow } A$ .
3. If  $\Lambda$  is in  $\text{First } \omega$  and  $\$$  is in  $\text{Follow } A$ , add  $A \rightarrow \omega$  to  $M[A,\$]$ .

$\text{First } E = \{ "(", "id" \}$

$\text{First } F = \{ "(", "id" \}$

$\text{First } T = \{ "(", "id" \}$

$\text{First } E' = \{ "+", \Lambda \}$

$\text{First } T' = \{ "*", \Lambda \}$

$\text{Follow } E = \{ ")", "\$" \}$

$\text{Follow } F = \{ "+", "*", ")", "\$" \}$

$\text{Follow } T = \{ "+", ")", "\$" \}$

$\text{Follow } E' = \{ ")", "\$" \}$

$\text{Follow } T' = \{ "+", ")", "\$" \}$

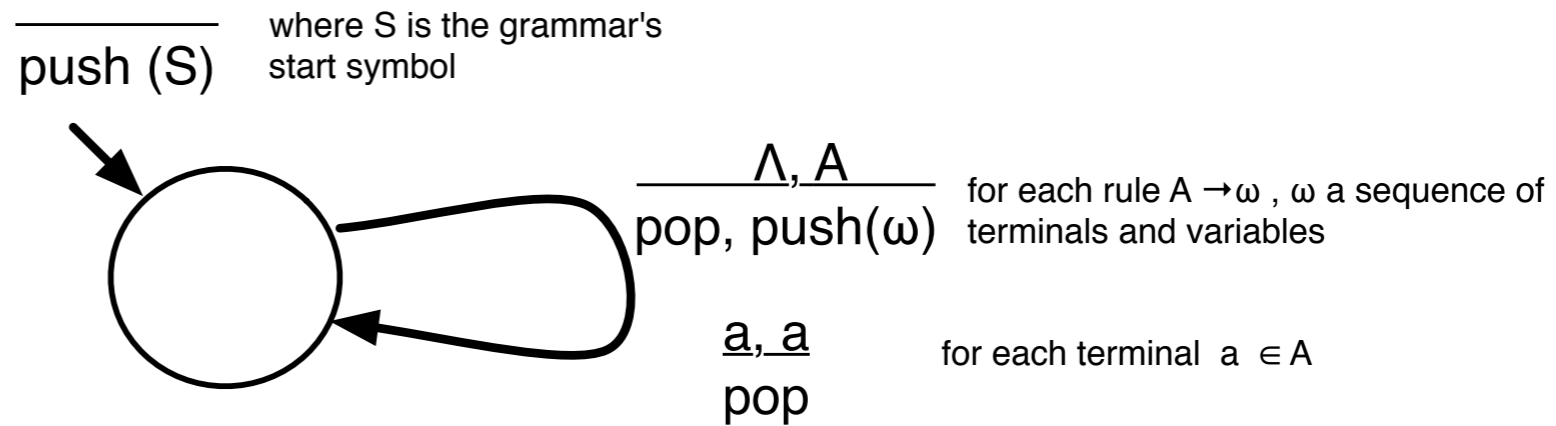
$M[A,t]$		terminals					
		+	*	)	(	id	\$
non-terminals	E				1	1	
	E'	2		3			3
	T				4	4	
	T'	6	5	6			6
	F				7	8	

1	E	→	T E' \$
2	E'	→	+ T E'
3			Λ
4	T	→	F T'
5	T'	→	* F T'
6			Λ
7	F	→	( E )
8			id

# Predictive Parsing Table

	id	+	*	(	)	\$
E	TE'			TE'		
E'		+TE'			Λ	Λ
T	FT'			FT'		
T'		Λ	*FT'		Λ	Λ
F	id			(E)		

# Table-driven Algorithm



push start symbol of grammar onto stack

repeat

let  $X$  = top of stack,  $c$  = next input

if terminal( $X$ )

then if  $X=c$

then pop  $X$ ; read  $c$

else error(...)

fi

else (\* nonterminal( $X$ ) \*)

if  $M[X, c] = Y_1 Y_2 \dots Y_k$

then pop  $X$ ;

push  $Y_k Y_{k-1} \dots Y_1$  (\*  $Y$  on top \*)

else error(...)

fi

fi

until stack is empty and input = \$

# Example Parse

Top of stack  
 ↓  
 Stack

E  
 T E'  
 F T' E'  
 id T' E'  
 T' E'  
 E'  
 + T E'  
 T E'  
 F T' E'  
 id T' E'  
 T' E'  
 E'

Input

x + y \$  
 x + y \$  
 x + y \$  
 x + y \$  
 + y \$  
 + y \$  
 + y \$  
 y \$  
 y \$  
 y \$  
 \$  
 \$  
 \$

	id	+	*	(	)	\$
E	T E'			T E'		
E'		+ T E'			Λ	Λ
T	F T'			F T'		
T'		Λ	* F T'		Λ	Λ
F	id			( E )		