CSL302: Compiler Design

Top Down Parsing - LL(1)

Vishwesh Jatala

Assistant Professor

Department of CSE

Indian Institute of Technology Bhilai

vishwesh@iitbhilai.ac.in

Acknowledgement

- Today's slides are modified from that of
  - *Stanford University:*
    - *https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/*

LL(1) Tables with ε

Num → Sign Digits
Sign → + | − | ε
Digits → Digit More
More → Digits | ε
Digit → 0 | 1 | … | 9

Num	Sign	Digit	Digits	More
+ − 0 5 1 6 2 7 3 8 4 9	+ − ε	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9	0 5 1 6 2 7 3 8 4 9 ε

	+	-	#	$
Num	Sign Digits	Sign Digits	Sign Digits	
Sign	+	-	ε	
Digits			Digits More	
More			Digits	ε
Digit			#	

The Final LL(1) Table Algorithm

- Compute FIRST(**A**) and FOLLOW(**A**) for all nonterminals **A**.
- For each rule $A \rightarrow \omega$, for each terminal $t \in \text{FIRST}(\omega)$, set $T[A, t] = \omega$.
  - Note that ε is not a terminal.
- For each rule $A \rightarrow \omega$, if $\varepsilon \in \text{FIRST}(\omega)$,
  set $T[A, t] = \omega$ for each $t \in \text{FOLLOW}(A)$.

A Formal Characterization of LL(1)

- A grammar is LL(1) if there are no conflicts in the table.
    - Every entry in the LL(1) table is unique

Exercise

- Construct the LL(1) parser table

  $S \rightarrow SA \mid b \mid \in$

  $A \rightarrow (A) \mid a$

Exercise

- Construct the LL(1) parser table

  $S \rightarrow SA \mid b \mid \in$

  $A \rightarrow (A) \mid a$

	FIRST	FOLLOW
S	{b, ∈ ,a,(}	{(,a,$}
A	{(,a}	{),(,a,$}

Exercise

- Construct the LL(1) parser table

  S → SA | b | ∈

  A → (A) | a

	a	b	(	)	$
S	**S->SA** **S->∈**	S->b	**S->SA** **S->∈**		S->∈
A	A->a		A->(A)		

Exercise

- Construct the LL(1) parser table

$$A \rightarrow Ab \mid c$$

A Grammar that is Not LL(1)

- Consider the following (left-recursive) grammar:

  $A \rightarrow Ab \mid c$

- FIRST(A) = {c}

	b	c
A		$A \rightarrow Ab$ $A \rightarrow c$

A Grammar that is Not LL(1)

- Consider the following (left-recursive) grammar:

  $A \to Ab \mid c$

- FIRST(A) = {c}

	b	c
A		$A \to Ab$ $A \to c$

- **Cannot uniquely predict production!**
- This is called a **FIRST/FIRST conflict**.

Eliminating Left Recursion

- In general, left recursion can be converted into **right recursion** by a mechanical transformation.
- Consider the grammar

$$A \to A\omega \mid a$$

- This will produce a followed by some number of ω's.
- Can rewrite the grammar as

$$A \to aB$$

$$B \to \varepsilon \mid \omega B$$

The Strengths of LL(1)

- LL(1) is straightforward
  - Can be implemented quickly with a table- driven design.
- LL(1) is Fast
  - Can parse in $O(n\ |G|)$ time, where n is the length of the string and $|G|$ is the size of the grammar.

Exercise

- Construct the LL(1) parser table

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid id$

Exercise

Non - terminal	Input Symbol					
	id	$+$	$*$	(	)	\$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to \mathbf{id}$			$F \to (E)$		

Exercises

- Text book:
  - Example 4.27
  - Example 4.29
  - Example 4.33

Summary

- **Top-down parsing** tries to derive the user's program from the start symbol.
- **Leftmost BFS** is one approach to top-down parsing; it is mostly of theoretical interest.
- **Leftmost DFS** is another approach to top-down parsing that is uncommon in practice.
- **LL(1)** parsing scans from left-to-right, using one token of lookahead to find a leftmost derivation.
  - **FIRST sets** contain terminals that may be the first symbol of a production.
  - **FOLLOW sets** contain terminals that may follow a nonterminal in a production.
  - **Left recursion** cause LL(1) to fail and can be mechanically eliminated in some cases.

Questions?