Compilers Design

Mohamed Taha BENNANI Assistant Professor, FST - El Manar University, URAPOP-FST

-Academic Year 2020-2021-

Outline

- LL(1) parsing principle
- First sets
  - Without ε
  - With ε
- Follow sets
  - Without ε
  - With ε
- Constructing parsing table
- Limits of LL(1)

Predictive Parsers

- They can predict which production to use
  - By looking the next few tokens
  - No backtracking
- Predictive parsers accept LL(K) grammars
  - L means "left-to-right" scan input
  - L means "leftmost derivation"
  - K means "predict based on k tokens of lookahead"
  - In practice, LL(1) is used

Predictive Parsers - LL(1)

- At each step, only one choice of production
- That is
  - When a non-terminal A is leftmost in a derivation
  - The next input symbol (Token) is t
  - There is a unique production A$\rightarrow \alpha$ to used
  - or no production to use (an error state)
- LL(1) is a recursive descent variant without backtracking

Definitions

Grammar:

S	$\rightarrow$	int
S	$\rightarrow$	(S OP S)
OP	$\rightarrow$	+
OP	$\rightarrow$	-

LL(1) Parsing Table:

	int	(	)	+	*
S	int	(S OP S)			
OP				+	*

Parsing Table

It contains as much lines as the non-terminals of the grammar. Each terminal has its own column. Each cell holds the production associated the non terminal of the such line and starts with the terminal of the column.An empty cell denotes a syntax error; it occurs when a non-terminal cannot derive a sentential using the corresponding terminal.

Parsing Example

S$	(int+(int*int))$
(S OP S)$	(int+(int*int))$
S OP S)$	int+(int*int))$
int OP S)$	int+(int*int))$
OP S)$	+(int*int))$
+ S)$	+(int*int))$
S)$	(int*int))$
(S OP S))$	(int*int))$

S OP S))$	int*int))$
int OP S))$	int*int))$
OP S))$	*int))$
*S))$	*int))$
S))$	int))$
int))$	int))$
))$	))$
$	$

Grammar:

(1): S → int
(2): S → (S OP S)
(3): OP → +
(4): OP → -

LL(1) Parsing Table:

	int	(	)	+	*
S	(1)	(2)			
OP				(3)	(4)

Error Detection

Empty stack and input not analyzed

S$	int+int$
int$	int+int$
$	+int$

Grammar:

(1): S → int
(2): S → (S OP S)
(3): OP → +
(4): OP → -

Empty cell: No production to reduce the non-terminal

S $	(int(int))$
(S OP S)$	(int(int))$
S OP S)$	int(int))$
int OP S)$	int(int))$
OP S)$	(int))$

LL(1) Parsing Table:

	int	(	)	+	*
S	(1)	(2)			
OP				(3)	(4)

Parsing Algorithm

Let S the start symbol of the grammar, T the related parsing table, and ω the input sentence to be analyzed.

- Initialize the stack with S$
- Repeat until empty stack
  - Let "t" the first lexeme of the input ω
  - If the top of the stack contains the terminal "r"
    - if "r" = "t", then remove "r" from the stack and "t" from ω
    - otherwise, return a syntax error
  - If the top of the stack contains the non-terminal "A"
    - if T[A,t] is a non empty cell, substitute the top of the stack by its related production
    - otherwise, return a syntax error

Computing First Sets - Without ε

Algorithm

- For each non-terminal A, itialize FIRST(A)={t/ $A \rightarrow t\omega$ }
- Repeat until the sets remain the same: For each non terminal A, and for every production following the pattern: $A \rightarrow B\omega$ update the set using: $FIRST(A) = FIRST(A) \cup FIRST(B)$

Example

(1):	S	$\rightarrow$	if EXP then S	(5):	EXP	$\rightarrow$	not EXP
(2):	S	$\rightarrow$	while EXP do S	(6):	EXP	$\rightarrow$	++id
(3):	S	$\rightarrow$	EXP	(7):	TER	$\rightarrow$	id
(4):	EXP	$\rightarrow$	TER $\rightarrow$ id	(8):	TER	$\rightarrow$	const

Computing First Sets - Constructing Parsing Table 1

Algorithm

- Compute the First Set for each non terminal of the grammar
- For each production $A \rightarrow t\omega$, put T[A,t]=$t\omega$
- for each production $A \rightarrow B\omega$, put T[A,t]=$B\omega$ for each $t \in FIRST(B)$

Example

	if	then	while	do	;	not	++	→	id	const
S	1		2			3	3		3	3
EXP						5	6		4	4
TER									7	8

Computing First Sets - With ε

Algorithm

- At the initialization, for each non-terminal A, initialize FIRST(A)=$\{t/\ A \rightarrow t\omega\ \}$
- For each non-terminal A having a production $A \rightarrow \varepsilon$, add ε to FIRST(A)
- Repeat until the sets remain the same:
  - $\forall$ production, $A \rightarrow \alpha$ where α is sentential and FIRST(α) contains ε, update the set following the rule:$FIRST(A) \cup = \{\varepsilon\}$
  - $\forall$ production, $A \rightarrow \alpha t\omega$, where $\varepsilon \in FIRST(\alpha)$, update the set following the rule: $FIRST(A) \cup = \{t\}$
  - $\forall$ production, $A \rightarrow \alpha B\omega$, where $\varepsilon \in FIRST(\alpha)$, update the set following the rule: $FIRST(A) \cup = (FIRST(B) \backslash \{\varepsilon\})$

Computing First Sets - With ε

Example

(1): S $\rightarrow$ GREET END
(2): GREET $\rightarrow$ hello
(3): GREET $\rightarrow$ welcome
(4): END $\rightarrow$ name
(5): END $\rightarrow$ ε

FIRST Sets

S	GREET	END
hello welcome	hello welcome	name ε

Parsing Table

	hello	welcome	name
S	(1)	(1)	
GREET	(2)	(3)	
END			(4)

Input Parsing

S$	hello name $
GEET END$	hello name $
hello END$	hello name $
END$	name $
name$	name $

Computing First Sets - With ε

Parsing Table

Add a new column and put ε in each cell related to non-terminals with ε production.

	hello	welcome	name	$
S	(1)	(1)		
GREET	(2)	(3)		
END			(4)	(5)

S$	welcome $
GEET END$	welcome $
welcome END$	welcome $
END$	$
$	$

Exercise

Compute the FIRST sets of the foolowing grammar:

$$
\begin{aligned}
S &\rightarrow A\ a \\
A &\rightarrow B\ D \\
B &\rightarrow b|\varepsilon \\
D &\rightarrow d|\varepsilon
\end{aligned}
$$

Definitions

- With ε-productions, we need, sometimes, to look beyond the next non-terminal. The set of the terminals which are located after the next non-terminal is called FOLLOW.
- FOLLOW(A) $= \{t/S \Rightarrow^* \alpha A t \omega$, for $\forall \alpha, \omega\}$, where S is the start symbol of the grammar.

Algorithm

- At the initialization, for each non-terminal A, initialize FOLLOW(A)=$\{$t/ $B \rightarrow \alpha A t \omega$ $\}$
- Add \$ to FOLLOW(S), where S is the start symbol
- Repeat until the sets remain the same:
  - If $B \rightarrow \alpha A \omega$ is a production, update the set following the rule:$FOLLOW(A) \cup = FIRST(\omega) \backslash \{\varepsilon\}$
  - If $B \rightarrow \alpha A \omega$ is a production, and $\varepsilon \in$ FIRST(ω), update the set following the rule: $FOLLOW(A) \cup = FOLLOW(B)$

Example

Grammar

$$
\begin{aligned}
S &\rightarrow aABb \\
A &\rightarrow aAc \mid \varepsilon \\
B &\rightarrow bB \mid c
\end{aligned}
$$

	FIRSTS
A	$\{\varepsilon$,a$\}$
B	{b,c}
S	{a}

	FOLLOWS
A	{b,c}
B	{b}
S	{$}

Parsing Table

	a	b	c	$
A	aAc	ε	ε	
B		bB	c	
S	aABb			

Parsing Example

Input: "aacbbcb"

Algorithm

- Compute the FIRST and FOLLOW Sets for each non terminal of the grammar
- For each production $A \rightarrow \omega$, for each t $\in$ FIRST(ω), put T[A,t]=ω
- For each production $A \rightarrow \omega$, if $\varepsilon \in$ FIRST(ω), put T[A,t]=ω for each $t \in FOLLOW(A)$

Property 1

A grammar is LL(1) if and only if for every non-terminal A and every strings of symbols α, β such that $\alpha \neq \beta$ and: $A \rightarrow \alpha|\beta$ we have:

1. FIRST(α) $\bigcap$ FIRST(β) $= \varnothing$
2. if $\alpha \Rightarrow^* \varepsilon$ then FIRST(β) $\bigcap$ FOLLOW(A) $= \varnothing$

Property 2

The complexity of the predictive parsing is: $O(n|G|)$, where n is the length of the input to be parsed and $|G|$ is the number of the productions od of the grammar

Exercise

Show that the following grammar is LL(1) despite isn't left-factorised $S \rightarrow Ax|A$
$A \rightarrow \varepsilon$