CPS 140 - Mathematical Foundations of CS
Dr. S. Rodger
Section: LL Parsing (handout)

LL(k) Parser:

- top-down parser - starts with start symbol on stack, and repeatedly replace nonterminals until string is generated.
- predictive parser - predict next rewrite rule
- first L of LL means - read input string left to right
- second L of LL means - produces leftmost derivation
- k - number of lookahead symbols

LL parsing process:

- convert CFG to PDA (different method than before)
- Use the PDA and lookahead symbols
- Lookahead symbol is next symbol in input string

Convert CFG to NPDA

Idea: To derive a string with a CFG, start with the start symbol and repeatedly apply production rules until the string is derived. In order to *simulate* this process with an NPDA, start by pushing the start symbol on the stack. Whenever a production rule A→w would be applied, the variable A should be on top of the stack. A is popped (or replaced) and the right hand side of the rule, w, is pushed onto the stack. Whenever a terminal is on top of the stack, if it matches the next symbol in the input string, then it is popped from the stack. If it does not match, then this string is not in the language of the grammar. If starting with the start symbol S, one can apply replacement rules, match all the terminals in the input string and empty the stack, then the string is in the language.

The constructed NPDA:

- Three states: s, q, f

  start in state s

  push S on stack, move into q

  all rewrite rules in state q: If lhs of rewrite rule on top of stack, replace it with rhs of rewrite rule and stay in state q

  additional rules in q to recognize nonterminals: read input symbol, pop input symbol, stay in state q

  pop z from stack, move into f, accept

Example:

$L = \{a^n bb^n : n \geq 0\}$ S → aSb | b

A parsing routine for this grammar:

symbol is the lookahead symbol and $ is the end of string marker.

```
state = s
push(S)
state = q
read(symbol)                                  obtain the lookahead symbol
while top-of-stack ≠ z do
   case top-of-stack of
      S: if symbol == a then
            {pop(); push(aSb)}                replace S by aSb
         else if symbol == b then
            {pop(); push(b)}                  replace S by b
         else error
      a: if symbol ≠ a, then error
            else {pop(); read(symbol)}        pop a, get next lookahead
      b: if symbol ≠ b, then error
            else {pop(); read(symbol)}        pop b, get next lookahead
      end case
end while
pop()                                         pop z from the stack
if symbol ≠ $ then error
state = f
```

LL Parse Table - 2-dim array

When the grammar is large, the parsing routine will have many cases. Alternatively, store the information of which rule to apply in a table.

- rows - variables
- cols - terminals, $ (end of string marker)
- LL[i,j] contains the rhs of a rule. This rhs is pushed onto the stack when the lhs of the rule is the variable representing the ith row and the lookahead is the symbol representing the jth column.

Example: Parse table for

$L = \{a^n bb^n : n \geq 0\}$

S $\rightarrow$ aSb | b

A generic parsing routine

Idea: To replace a variable on the top of the stack with its appropriate rhs, use the lookahead and the lhs to look up the rhs in the LL parse table. (LL[,] is the parse table.

```
push(S)
read(symbol)                                  obtain the lookahead symbol
while stack not empty do
   case top-of-stack of
      terminal:
         if top-of-stack == symbol
            then {pop(); read(symbol)}        pop terminal and get next lookahead
         else
            error
      variable:
         if LL[top-of-stack, symbol] ≠ error
            then {pop()                       pop the lhs
               push(LL[top-of-stack,symbol])} push the rhs
         else
            error
   end case
end while
if symbol ≠ $, then error
```

For previous example, try the following traces:

Parse the string: aabbb

Parse the string: b

Example:

$S \rightarrow aSb$
$S \rightarrow c$

	a	b	c	$
S	aSb	error	c	error

In this example, it is clear that when S is on the stack and a is the lookahead, replace S by aSb. When S is on the stack and b is the lookahead, there is an error because there must be a c between the a's and b's. When S is on the stack and $ is the lookahead, then there is an error since S must be replaced by at least one terminal. When S is on the stack, and c is the lookahead, then S should be replaced by c.

Example:

$S \rightarrow Ac \mid Bc$
$A \rightarrow aAb \mid \epsilon$
$B \rightarrow b$

When the grammar has a ϵ-rule, it can be difficult to compute parse tables. In the example above, A can disappear (due to $A \rightarrow \epsilon$), so when S is on the stack, it can be replaced by Ac if either "a" or "c" are the lookahead or it can be replaced by Bc if "b" is the lookahead.

We will use the following functions FIRST and FOLLOW to aid in computing the table.

To construct an LL parse table LL[rows,cols]:

1. For each rule $A \rightarrow w$
   (a) For each a in FIRST(w)
   add w to LL[A,a]
   (b) If ϵ is in FIRST(w)
   add w to LL[A,b] for each b in FOLLOW(A)
   where $b \in T \cup \{\$\}$
2. Each undefined entry is error.

Example:

$$S \rightarrow aSc \mid B$$
$$B \rightarrow b \mid \epsilon$$

We have already calculated FIRST and FOLLOW for this Grammar:

	FIRST	FOLLOW
S	a,b,ϵ	\$,c
B	b,ϵ	\$,c

To Compute the LL Parse Table for this example:

- For S $\rightarrow$ aSc,
  FIRST(aSc) =
- For S $\rightarrow$ B,
  FIRST(B) = {b, ϵ}
  FOLLOW(S) = {\$, c}
- For B $\rightarrow$ b,
  FIRST(b) =
- For B $\rightarrow \epsilon$
  FIRST(ϵ) =

LL(1) Parse Table:

	a	b	c	\$

Parse string: aacc

Another trace example:

Trace aabcc

				a					
		a		S	S	B	b		
		S	S	c	c	c	c	c	
Stack:	S	c	c	c	c	c	c	c	c
symbol:	a	a	a'	a'	b	b	b	c	c'

where a' is the second a in the string and symbol is the lookahead symbol. This table is an LL(1) table because only 1 symbol of lookahead is needed.

Example: Construct Parse Table for:

$L = \{a^n b^n c a^m c b^m : n \geq 0, m \geq 0\}$

S $\rightarrow$ AcB
A $\rightarrow$ aAb
A $\rightarrow \epsilon$
B $\rightarrow$ aBb
B $\rightarrow$ c

FIRST(A) =

FIRST(S) =

FIRST(B) =

FOLLOW(A) =

FOLLOW(S) =

FOLLOW(B) =

To compute the parse table:

- For S $\rightarrow$ AcB,
  FIRST(AcB) =
- For A $\rightarrow$ aAb,
  FIRST(aAb) =
- For A $\rightarrow \epsilon$,
  FIRST(ϵ) =
- For B $\rightarrow$ aBb,
  FIRST(aBb) =
- For B $\rightarrow$ c,
  FIRST(c) =
- All other entries are errors.

LL(1) Parse Table:

	a	b	c	$

parse string: abcacb

parse string: cc

parse string: abcab (not in language)

Example:

S → AcB
A → aAb
A → ab
B → aBb
B → acb

	FIRST	FOLLOW

Try to construct LL(1) Parse table

	a	b	c	$

LL(2) Parse Table:

	aa	ab	ac	a$	b	c	$
S	AcB	AcB	error	error	error	error	error
A	aAb	ab	error	error	error	error	error
B	aBb	error	acb	error	error	error	error

parse string: aabbcacb

			a		a							
			A	A	b	b						
		A	b	b	b	b	b			a		
		c	c	c	c	c	c	c		c	c	
Stack:	S	B	B	B	B	B	B	B	B	b	b	b
symbol:	aa	aa	aa	ab	ab	bb	bc	ca	ac	ac	cb	b$

Example: $L = \{a^n : n \geq 0\} \cup \{a^n b^n : n \geq 0\}$

S $\rightarrow$ A
S $\rightarrow$ B
A $\rightarrow$ aA
A $\rightarrow \epsilon$
B $\rightarrow$ aBb
B $\rightarrow \epsilon$

Example: $L = \{a^n : 0 \leq n \leq 10\} \cup \{a^n b^n : 0 \leq n \leq 10\}$

Example: $L = \{a^n b b^n : n \geq 0\} \cup \{a^n b^n : n \geq 0\}$

S $\rightarrow$ aSb
S $\rightarrow$ b
S $\rightarrow \epsilon$