

Exercises on Grammars

1. Consider the following grammar:

$S \rightarrow (L) \mid a$

$L \rightarrow L , S \mid S$

- Is this grammar ambiguous ?
- Is this grammar LL(1) ?
- Compute the First and Follow sets for the new grammar.
- Construct the parsing table for the LL(1) parser

Finding an LL(1) grammar

- No procedural way ! Practice ...
- But there are some recommended practices that generally help in finding one.
- Eg. try to eliminate left recursion.
 - There is a procedure for this but you don't have to faithfully follow the entire approach.
 - Just think of what left recursion brings and what can be done to eliminate them

Removing Left Recursion

$S \rightarrow (L) \mid a$

$L \rightarrow L, S \mid S$

- How does a derivation starting from 'L' look ?

- $L \Rightarrow L, S$

$\Rightarrow L, S, S$

$\Rightarrow^* L, S, \dots, S$

$\Rightarrow S, \dots, S$

- $L \rightarrow L, S \mid S$ is equivalent to $L \rightarrow S, L \mid S$

$S \rightarrow (L) \mid a$

$L \rightarrow S, L \mid S$

Removing Left Recursion

- In general, $L \rightarrow L \alpha \mid \beta_1 \mid \dots \mid \beta_n$
- $L \rightarrow \beta_1 Z \mid \dots \mid \beta_n Z \mid \beta_1 \mid \dots \mid \beta_n$
- $Z \rightarrow \alpha Z \mid \epsilon$
- This will remove immediate recursion but only when there are no epsilon productions in the grammar
- Otherwise, we need to remove epsilon productions which will be discussed along with CYK parsing
- Removing indirect recursion

$S \rightarrow L a$

$L \rightarrow S a \mid b$

Removing Left Recursion

- Order nonterminals Eg. (1) S , (2) L
- Enforce that if $A \rightarrow B$ then A should precede B in the ordering
- $S \rightarrow L a$ and $L \rightarrow b$ satisfy the constraint but $L \rightarrow S a$ doesn't
- Inline the production of S in $L \rightarrow S a$
- We get, $L \rightarrow L a a \mid b$, Remove left recursion.
 - Result: $L \rightarrow b Z \mid b$ $Z \rightarrow a a Z \mid \epsilon$
- If inlining does not result in left recursive production or doesn't satisfy the constraints, **inline again.**

Example 1 [Cont.]

$S \rightarrow (L) \mid a$

$L \rightarrow L , S \mid S$

- After eliminating left recursion

$S \rightarrow (L) \mid a$

$L \rightarrow S , L \mid S$

- Is this LL(1) now ?

Example 1 [Cont.]

$S \rightarrow (L) \mid a$

$L \rightarrow L , S \mid S$

- After eliminating left recursion

$S \rightarrow (L) \mid a$

$L \rightarrow S , L \mid S$

- Is this LL(1) now ?

Left factorization

$S \rightarrow (L) \mid a$

$L \rightarrow S , L \mid S$

- Identify a common prefix and push the suffixes to a new nonterminal.

$S \rightarrow (L) \mid a$

$L \rightarrow S Z$

$Z \rightarrow , L \mid \epsilon$

- Is this LL(1) now ? **Yes**

Exercise 1 - First and Follow sets (with EOF)

$S \rightarrow (L) \text{ EOF } \mid a \text{ EOF}$

$L \rightarrow S Z$

$Z \rightarrow , L \mid \epsilon$

- $First(S) \supseteq First((L)) \cup First(a) = \{ (, a \}$
- $First(L) \supseteq First(S Z) = First(S)$
- $First(Z) \supseteq First(, L) = \{ , \}$
- $Follow(S) \supseteq Follow(L) \cup Follow(Z)$
- $Follow(L) \supseteq \{) \} \cup Follow(Z)$
- $Follow(Z) \supseteq Follow(L)$

First and Follow sets [Cont.]

$S \rightarrow (L) \text{ EOF } \mid a \text{ EOF}$

$L \rightarrow S Z$

$Z \rightarrow , L \mid \epsilon$

- Solution to the above constraints:
 - $First(S) = First(L) = \{ (, a \}$
 - $First(Z) = \{ , \}$
 - $Follow(S) = Follow(L) = Follow(Z) = \{) \}$
- Moreover, Z is Nullable

LL(1) parsing table

(1) $S \rightarrow (L) EOF$

(2) $S \rightarrow a EOF$

(3) $L \rightarrow S Z$

(4) $Z \rightarrow , L$

(5) $Z \rightarrow \epsilon$

	a	()	,	EOF
S	2	1	Error	Error	Error
L	3	3	Error	Error	Error
Z	Error	Error	5	4	Error

Exercise 2

Consider a grammar for expressions where the multiplication sign is optional.

$ex ::= ex + ex \mid ex * ex \mid ex \ ex \mid ID$

- Find a LL(1) grammar recognizing the same language
- Create the LL(1) parsing table.

Exercise 2 – Solution

- First let's make the grammar unambiguous by associating precedence with operators
- In the process we also made sure that the grammar does not have left recursion
- $ex ::= S + ex \mid S$
- $S ::= ID * S \mid ID S \mid ID$
- Left factorization:
- $ex ::= S Z$
- $Z ::= + ex \mid \epsilon$
- $S ::= ID Z_2$
- $Z_2 ::= * S \mid S \mid \epsilon$

Exercise 2 – LL(1) parsing table

- $ex ::= S Z EOF$
- $Z ::= + ex \mid \epsilon$
- $S ::= ID Z2$
- $Z2 ::= * S \mid S \mid \epsilon$
- First let's compute first and follow sets after adding EOF to the end of the start symbol productions
 - $First(ex) = First(S) = \{ ID \}$
 - $First(Z) = \{ + \}$ $First(Z2) = \{ * , ID \}$
 - $Follow(ex) = Follow(Z) = \{ EOF \}$
 - $Follow(S) = Follow(Z2) = \{ EOF, + \}$
- Z and Z2 are nullable

LL(1) parsing table

1. $ex ::= S Z EOF$
2. $Z ::= + ex$
3. $Z ::= \epsilon$
4. $S ::= ID Z2$
5. $Z2 ::= * S$
6. $Z2 ::= S$
7. $Z2 ::= \epsilon$

	ID	+	*	EOF
ex	1	Error	Error	Error
Z	Error	2	Error	3
S	4	Error	Error	Error
Z2	6	7	5	7

Exercise 3

Balanced Parentheses over $\{ (, [\}$

$S ::= (S) \mid [S] \mid S S \mid \epsilon$

- Find a LL(1) grammar recognizing the language

Exercise 3 - Solution

- $S ::= (S) | [S] | S S | \epsilon$
- 'S' produces epsilon. Hence, we need to first eliminate epsilon (discussed later) and then remove left recursion from $S ::= S S$
- Instead, let's apply the same logic as removing left recursion but without performing all the steps.
- The role of the production $S ::= S S$ is to produce a sequence of S that begin with either (S) or [S]. i.e.,
 - (S) S S S
 - [S] S S..... S

Exercise 3 - Solution

- Each of the successive S 'es can rewrite to either (S) or $[S]$. That is, in essence $S ::= S S$ produces sequences given by the regular expression $((S) | [S])^*$
 - E.g $(S) (S) [S] (S) \dots$ is one such sequence
- The same effect can be achieved by the right recursive rules
 - $S ::= (S) S | [S] S | \epsilon$
- The above grammar is LL(1)

Exercise 4

Prove that every LL(1) grammar is unambiguous.

Solution to Exercise 4

Intuition:

Every production of a non-terminal belonging to an LL(1) grammar generates a set of strings that is completely disjoint from the other alternatives because of the following two reasons:

- (a) For every nonterminal, the first sets of every alternative are disjoint which implies that they produce disjoint non-empty strings.
- (b) There is at most one production for a non-terminal that can produce an empty string

Formal proof is presented in the next slide

Solution to Exercise 4 [Cont.]

Claim : Every string w derivable from every non-terminal N belonging to an LL(1) grammar in n -steps has a unique left most derivation.

Base case: w is derivable from N in 1 step.

- In this case, there has to exist a production $N \rightarrow w$
- Say w is non-empty i.e, $w = a x$
 - Clearly, there cannot be any other production of N that can derive w . Otherwise, its first set will intersect with that of the the production $N \rightarrow w$
- Say w is empty
 - Clearly, there cannot be any other production of N that can derive empty string. Otherwise, N will have two nullable productions.

Solution to Exercise 4 [Cont.]

Inductive case: w is derivable from N in k steps i.e, $N \Rightarrow^k w$

- Say w is non-empty i.e, $w = a x$
 - For every nonterminal N , s.t $N \Rightarrow^* w$, the first rule that N uses is the alternative of N whose first set has 'a'
 - Let the alternative of N whose first set has 'a' be $N \rightarrow \alpha$
 - $N \Rightarrow A_1 A_2 \cdots A_n \Rightarrow^{k-1} w$
 - Let w_i be the substring of w generated by A_i
 - Clearly, A_i generates w_i in less than k steps i.e, $A_i \Rightarrow^j w_i$, $j \leq k - 1$
 - By hypothesis, $A_i \Rightarrow^j w_i$ is unique for each A_i . Hence, $N \Rightarrow w$ is also unique
- Say w is empty
 - Similar to the above argument, the first rule that N uses is the alternative of N that is nullable which is say $N \rightarrow \alpha$
 - The rest of the proof is similar to the above case

Exercise 5

Say that a grammar has a cycle if there is a *reachable, productive* non-terminal A such that $A \Rightarrow^* A$, i.e. it is possible to derive the nonterminal A from A by a nonempty sequence of production rules.

Show that if a grammar has a cycle, then it is not LL(1).

(the solution is provided in a separate pdf file in the lara wiki)

Exercise 6

Show that the regular languages can be recognized with LL(1) parsers. Describe a process that, given a regular expression, constructs an LL(1) parser for it.

Solution for Exercise 6

- Let the DFA for the regular language be $A : (\Sigma, Q, q_0, \delta, F)$
- Define a grammar $G: (N, T, P, S)$ where,
- $N = \{ S_i \mid 1 \leq i \leq |Q| \}$
- $T = \Sigma$
- $S = S_0$
- $\delta(q_i, a) = q_j \Rightarrow S_i \rightarrow a S_j \in P$
- $q_i \in F \Rightarrow S_i \rightarrow \epsilon \in P$

$$L(A) = L(G)$$

Exercise 7

Show that the language $\{ a^n b^m \mid n > m \}$ cannot have an LL(1) grammar ?

Note that the following grammar recognizes the language but is not LL(1)

$$S \rightarrow a S \mid P$$
$$P \rightarrow a P b \mid a$$

This question interesting but is quite difficult. A proof for this is provided in a separate pdf file in the lara wiki.

This is meant only as a supplementary material to provide more insights into LL(1) grammars.

It is not essential to fully understand the proof of this question.