Exercises on Grammars

1. Consider the following grammar:

S -> (L) | a

L -> L , S | 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 -> (L) | a

L -> L , S | S

- How does a derivation starting from 'L' look ?
- L => L , S

  => L , S , S

  =>* L , S , … , S

  => S , … , S
- L -> L , S | S is equivalent to L -> S , L | S

  S -> (L) | a

  L -> S , L | S

Removing Left Recursion

- In general, L -> L α | β_1 | … | β_n
- L -> β_1 Z | … | β_n Z | β_1 | … | β_n
- Z -> α Z | ϵ
- 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 discussed along with CYK parsing
- Removing indirect recursion

  S -> L a

  L -> S a | b

Removing Left Recursion

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

Example 1 [Cont.]

S -> (L) | a

L -> L , S | S

- After eliminating left recursion

S -> (L) | a

L -> S , L | S

- Is this LL(1) now ?

Example 1 [Cont.]

S -> (L) | a

L -> L , S | S

- After eliminating left recursion

S -> (L) | a

L -> S , L | S

- Is this LL(1) now ?

Left factorization

S -> (L) | a

L -> S , L | S

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

S -> (L) | a

L -> S Z

Z -> , L | ϵ

- Is this LL(1) now ? Yes

Exercise 1 - First and Follow sets (with EOF)

S -> (L) EOF | a EOF

L -> S Z

Z -> , L | ϵ

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

First and Follow sets [Cont.]

S -> (L) EOF | a EOF

L -> S Z

Z -> , L | ϵ

- 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 -> (L) EOF
(2) S -> a EOF
(3) L -> S Z
(4) Z -> , L
(5) Z -> ϵ

	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 | ex * ex | ex ex |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 | S
- S ::= ID * S | ID S | ID
- Left factorization:
- ex ::= S Z
- Z ::= + ex | ϵ
- S ::= ID Z2
- Z2 ::= * S | S | ϵ

Exercise 2 – LL(1) parsing table

- ex ::= S Z EOF
- Z ::= + ex | ϵ
- S ::= ID Z2
- Z2 ::= * S | S | ϵ
- 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 ::= ϵ
4. S ::= ID Z2
5. Z2 ::= * S
6. Z2 ::= S
7. Z2 ::= ϵ

	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)| [S] | S S | ϵ

- Find a LL(1) grammar recognizing the language

Exercise 3 - Solution

- $S ::= (S) \mid [S] \mid S S \mid \epsilon$
- '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) … is one such sequence
- The same effect can be achieved by the right recursive rules
  - S ::= (S) S | [S] S | ϵ
- 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 -> 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 -> 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 =>* 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 → $A_1 A_2 \cdots A_n$
  - $N \Rightarrow A_1 A_2 \cdots A_n \Rightarrow^{k-1} \; w$
  - 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, it suffices to show that there is no other partition $u_1 \cdots u_n$ of w such that $A_i \Rightarrow^{<k} \; u_i$
  - Let j be the first substring such that $u_j \neq w_j$ .Given $A_j \Rightarrow^* \; u_j$ and $A_j \Rightarrow^* \; w_j$
  - Let $u_j = xy_1$ and $w_j = xy_2$, $A \Rightarrow^* \; x\,\alpha \Rightarrow^* \; xy_1$, $A \Rightarrow^* \; x\,\alpha \Rightarrow^* \; xy_2$
  - Both y_1 and y_2 cannot be empty otherwise u_j and w_j are equal.

Solution to Exercise 4 [Cont.]

- Let ‘a’ be the start of y_1 or y_2 whichever is nonempty
- α cannot be empty as it has to derive a. Let N be the first symbol of α
- N has two different productions on the same input symbol ‘a’ which is not possible in an LL(1) grammar.
- Hence, $u_j = w_j$

- 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 $\mathrm{N} \rightarrow A_1 A_2 \cdots A_n$
  - Each of the A_i’s should derive ϵ through a unique production by hypothesis.
  - Hence, the claim

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).

Solution to Exercise 5

- We proved before that LL(1) grammars are not ambiguous
- Consider a left most derivation D that contains A
- D: $S \Rightarrow^* \ xA\beta \Rightarrow^* \ w$
  - Where, x is a (possibly empty) sequence of terminals and
  - β is a sentential form
  - Such a derivation must exist as A is reachable (from the start symbol) and also productive
- Using A $\Rightarrow^+$A, we can derive another derivation for w
- D′: $S \Rightarrow^* \ xA\beta \Rightarrow^+ xA\beta \Rightarrow^* \ w$
- There exists two left most derivations and hence two parse trees for w
- The grammar is ambiguous and hence cannot be LL(1)

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 = $\{ S_i \mid 1 \leq i \leq |Q| \}$
- $\mathrm{T} = \Sigma$
- 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 -> a S | P

P -> a P b | 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.