Exercises on Chomsky Normal Form and CYK parsing

1. Convert the following grammar to CNF

S -> A (S) B | “”

A -> S | S B | x | “”

B -> S B | y

This is exercise is available in the “grammar tutoring system”

http://laraserver3.epfl.ch:9000

- Select exercise type “CNF Conversion” -> choose the problem “Exercise 1 of lecturecise 12”
- Make sure you create a new start symbol S1 and add the production S1 -> S | “” to your grammar before CNF conversion as the start symbol ‘S’ is nullable and also appears on the right hand side

Exercise 2

Which of the following properties of a grammar are preserved by the "Epsilon Production Elimination" algorithm

a. Ambiguity No, counter-example ?

   If a grammar is "ambiguous" does it remain ambiguous after removing epsilon productions ?

b. LL(1) No, counter-example ?

c. No Left Recursion No, counter-example ?

   We define left recursion as existence of productions of the form

   – N -> N α (or)

   – $N \rightarrow N_1\, \alpha_1\,,\; N_1 \rightarrow N_2\, \alpha_2,\; \cdots, N_n \rightarrow \mathrm{N}\, \alpha_n$

d. Unambiguity ? Yes, proof ?

Exercise 2(a) - Solution

- Ambiguity : If a grammar is “ambiguous” does it remain ambiguous after removing epsilon productions ?
- No, it need not. Eg. consider the following ambiguous grammar
  - S -> a A | a
  - A -> b | “”
  - The grammar has two parse trees for the string: a
- After removing epsilon productions we get
  - S -> a A | a
  - A -> b
  - There is exactly one parse tree for “a“

Exercise 2(b) - Solution

- LL(1): If a grammar is LL(1) does it remain LL(1) after removing epsilon productions ?
- No, it need not. Eg. consider the following LL(1) grammar
  - S -> a S b | “”
- After removing epsilon productions we get
  - S’ -> S | “”
  - S -> a S b | a b
  - Note: we have created a new start symbol as ‘S’ was nullable and appeared on the right hand side
  - The grammar is not LL(1) as First(a S b) and First(a b) intersect

Exercise 2(c) - Solution

- No left recursion: If a grammar has no left recursion does it remain without left recursion after removing epsilon productions ?
- No, it need not. Eg. consider the following non-left recursive grammar
  - S -> B S a |a
  - B -> “” | b
- After removing epsilon productions we get
  - S -> B S a | S a | a
  - B -> “” | b
  - The production S -> S a is a left recursive production
- Note: this also means that we have to eliminate epsilons before removing left recursion using the approach described in lecturecise 10

Exercise 2(d) - Solution

Unambiguity: If a grammar is unambiguous does it remain unambiguous after removing epsilon productions ?

- Yes. Proof: Let G' be obtained from G by eliminating epsilon productions

Let's prove the contra-positive form: if there are two left most derivations for a word w in G' then there will be two left most derivations for w in G

  - Let D1 and D2 be two left most derivations for the word w in G'
  - If all the productions used in D1 and D2 are also present in the G then D1 and D2 are also feasible in G, which will imply the claim.
  - Therefore, say D1 or D2 use productions denoted using: $A_i \rightarrow \gamma_j$ that did not belong to G.
  - For each $A_i \rightarrow \gamma_j$ (except when A_i is a start symbol), there exists $A_i \rightarrow \alpha_j$ belonging to G such that γ_j is obtained from α_j by removing nullable non-terminals from some positions (denoted by say $\overline{X_j}$).
  - Hence, we can create new derivations D1' and D2' from D1 and D2 by replacing every use of the rule $A_i \rightarrow \gamma_j$ by $A \rightarrow \alpha_j$ and deriving empty strings (using productions of G) from non-terminals in positions $\overline{X_j}$

Exercise 2(d) - Solution

- If A_i was the start symbol of G' and it does not belong to G then we remove the derivation $A_i \rightarrow S$ at the start of G' and replace it with the start of symbol S of G
- Will the resulting derivations D1' and D2' be distinct ?
- Consider the point where D1 and D2 diverge. Let
  - D1: $S \Rightarrow^* xB\alpha \Rightarrow x\beta_1\alpha \Rightarrow^* w$
  - D2: $S \Rightarrow^* xB\alpha \Rightarrow x\beta_2\alpha \Rightarrow^* w$
- If neither $B \rightarrow \beta_1$ or $B \rightarrow \beta_2$ are of the form $A_i \rightarrow \gamma_j$ then D1' and D2' will also diverge at the same point as we preserve $B \rightarrow \beta_1$ and $B \rightarrow \beta_2$ in D1', D2' respec.
- Case (i), say
  - D1: $S \Rightarrow^* xA_i\alpha \Rightarrow x\gamma_j\alpha \Rightarrow^* w$
  - D2: $S \Rightarrow^* xA_i\alpha \Rightarrow x\beta\alpha \Rightarrow^* w$, where $A_i \rightarrow \beta$ is a rule in G as well
- γ_j is replaced by α_j in D1' and some non-terminals in α_j derive epsilon. However, no non-terminal in β derives epsilon in D2 as D2 is a derivation of G' that has no epsilon productions except for the start symbol. By construction, the start symbol of G' will never appear on the right hand side if it is nullable.
- Hence, $A_i \rightarrow \beta$ will be preserved in D2' and no non-terminal in β derives epsilon in D2'. Hence, D1' will differ from D2' irrespective of whether $\alpha_j = \beta$ or not.

Exercise 2(d) - Solution

- Case (ii), say
  - D1: $S \Rightarrow^* xA_i\alpha \Rightarrow x\gamma_j\alpha \Rightarrow^* w$
  - D2: $S \Rightarrow^* xA_i\alpha \Rightarrow x\gamma_k\alpha \Rightarrow^* w$
- γ_j is replaced by α_j in D1’ and γ_k by α_k in D2’. Since $\gamma_j \neq \gamma_k$ either α_j and α_k are different or non-terminals at different positions in α_j and α_k are reduced to empty string in D1’ and D2’, respectively. Hence, D1’ will differ from D2’

Exercise 3

Which of the following properties of a grammar are preserved by the "Unit Production Elimination" algorithm

- Ambiguity No, counter-example ?
- Left Recursion No
  - What about these rules: B -> A | a , A -> B ?
- LL(1) Yes, proof ?
- Unambiguity Yes, proof ?

Exercise 3(a) - Solution

- Ambiguity : If a grammar is “ambiguous” does it remain ambiguous after removing unit productions ?
- No, it need not. Eg. consider the following ambiguous grammar
  - S -> A | a
  - A -> a
  - There exists two parse tree for a: S -> A -> a and S -> a
- After removing the unit production we get
  - S -> a
  - There is exactly one parse tree for “a“

Exercise 3(b) - Solution

- Left recursion: If a grammar has left recursion will it have left recursion after removing unit productions ?
- No, it need not. Eg. consider the following left recursive grammar
  - B -> A | a
  - A -> B
- After removing the unit productions (using the graph based algorithm described in lecturecise 11) we get
  - B -> a
  - A -> a

Exercise 3(c) - Solution

- LL(1): If a grammar is LL(1) does it remain LL(1) after removing unit productions ?
- Yes. The following is a sketch of the proof (not a complete, rigorous proof).
- Let G’ be obtained from G by eliminating unit productions, we need show that all 3 properties of LL(1) holds
- Without loss of generality assume that we are removing only one unit production A -> B and B -> $\beta_1 \mid \cdots \mid \beta_n$
- A -> B will be replaced by A -> $\beta_1 \mid \cdots \mid \beta_n$
  1. Every alternative of A in G’ will have disjoint first sets. If there is an alternative whose first set intersects with β_i for some i then in G it would have intersected with first(B) contradicting the fact that the input grammar is LL(1)
  2. A will have at most one nullable alternative in G’. If in G, B was A’s nullable alternative then there will exist exactly one β_i that is nullable in G (and hence in G’) as G is in LL(1). If not, no β_i will be nullable in G (and hence in G’).

Exercise 3(c) – Solution [Cont.]

3. The follow set of every non-terminal except ‘B’ will be the same in G and G’. Note that the follow set of non-terminals in β_i cannot change by adding the production A -> β_1 |⋯ | β_n as in both G and G’, $follow(A) \subseteq follow(\beta_i)$ for each i, and G and G’ have same set of nullable non-terminals.

For the non-terminal B, follow(B) in G’ is a subset of follow(B) in G as the production A -> B is removed. However, any reduction in the follow set cannot violate LL(1) property. Hence, G’ is also in LL(1)

Exercise 3(d) - Solution

Unambiguity: If a grammar is unambiguous does it remain unambiguous after removing unit productions ?

- Yes. Sketch of the proof: Let G' be obtained from G by eliminating epsilon productions
- Let's prove the contra-positive form: if there are two left most derivations for a word w in G' then there will be two left most derivations for w in G
  - Without loss of generality assume that we are removing only one unit production A -> B and B -> β_1 $|\cdots|$ β_n . A -> B will be replaced by A -> β_1 $|\cdots|$ β_n
  - Let D1 and D2 be two left most derivations for the word w in G'
  - If all the productions used in D1 and D2 are also present in the G then D1 and D2 are also feasible in G, which will imply the claim.
  - Therefore, say D1 or D2 use productions denoted using: $A \rightarrow \beta_i$ that did not belong to G.
  - For each $A \rightarrow \beta_i$ there exists a derivation $A \rightarrow B \rightarrow \beta_i$ belonging to G we can create new derivations D1' and D2' from D1 and D2 by replacing every use of the rule $A \rightarrow \beta_i$ by the derivation $A \rightarrow B \rightarrow \beta_i$
  - Are D1' and D2' distinct ?

Exercise 3(d) - Solution [Cont.]

- Consider the point where D1 and D2 diverge. Let
  - D1: $S \Rightarrow^* \ xC\alpha \Rightarrow x\gamma_1\alpha \Rightarrow^* \ w$
  - D2: $S \Rightarrow^* xC\alpha \Rightarrow x\gamma_2\alpha \Rightarrow^* \ w$
- If C is not A or neither of γ_1 and γ_2 are β_i for some i then D1' and D2' will also diverge at the same point as we do not change such productions
- Case (i), say
  - D1: $S \Rightarrow^* \ xA\alpha \Rightarrow x\beta_i\alpha \Rightarrow^* \ w$
  - D2: $S \Rightarrow^* xA\alpha \Rightarrow x\gamma_2\alpha \Rightarrow^* \ w$
- In this case D1' will be $S \Rightarrow^* \ xA\alpha \Rightarrow xB\alpha \Rightarrow \ x\beta_i\alpha \Rightarrow^* \ w$ and D2' will have $S \Rightarrow^* xA\alpha \Rightarrow x\gamma_2\alpha$. Moreover, γ_2 is not B as A -> B does not belong to G'. Hence, D1' will differ from D2'
- Case (ii), say
  - D1: $S \Rightarrow^* \ xA_i\alpha \Rightarrow x\beta_i\alpha \Rightarrow^* \ w$
  - D2: $S \Rightarrow^* xA_i\alpha \Rightarrow x\beta_j\alpha \Rightarrow^* \ w$
- In this case D1': $S \Rightarrow^* \ xA\alpha \Rightarrow xB\alpha \Rightarrow \ x\beta_i\alpha \Rightarrow^* \ w$ and D2': $S \Rightarrow^* \ xA\alpha \Rightarrow xB\alpha \Rightarrow$ $x\beta_j\alpha \Rightarrow^* \ w$. Hence, D1' will differ from D2'

Exercise 4

Given a grammar G in CNF, how many steps does it require to derive a string of size n.

Exercise 4 -Solution

Intuition

Consider a derivation of a string of length n obtained as follows:

1. Derive a string of exactly n nonterminals from the start symbol, then
2. Expand each nonterminal out to a single terminal.

- To obtain 'n' non-terminals from the start symbol, we need to apply productions of the form S → AB as that is the only way to generate non-terminals. How many times do we have to apply such productions ?
- Application of one such production will increase the number of nonterminals by 1, since you replace one nonterminal with two nonterminals.
- Since we start with one nonterminal, we need to repeat this n-1 times.
- We need n more steps to convert nonterminals to terminals
- Therefore, total number of steps = 2n – 1
- Let's try to prove this bound formally

Exercise 4 –Solution [Cont.]

- Denote the number of steps required to derive a string w from a non-terminal N as NS(N, w)
- If |w| = 1, we need exactly 1 step. NS(N, w) = 1 if |w| = 1
- If |w| > 1, we first apply a production N -> N1 N2 which will derive w.
- Say $N \Rightarrow^* \; N1\, N2 \Rightarrow^* \; x\, N2 \Rightarrow^* \; xy = w$,
  - where |x| = q (say) and |y| = |w| - q
- Hence, NS(N,w) = NS(N1, x) + NS(N2, y) + 1
- We need a closed form for NS(N,w) that depends only on |w|. Say NS(N,w) = f(|w|)
- We have, $f(|w|) = \begin{cases} 1 \quad if\ |w| = 1 \\ f(q) + f(|w| - q) + 1 \ \ if\ |w| > 1 \end{cases}$
- Where 1 <= q < |w|
- $2|w| - 1$ is the least solution for the above recurrence

Exercise 5

Assume a grammar in CNF has *n* non-terminals. Show that if the grammar can generate a word with a derivation having at least 2^n steps, then the recognized language should be infinite

(see the pdf file uploaded in the lara wiki along with the slides)

Exercise 6

Show the CYK parsing table for the string "aabbbab"

for the grammar

S -> AB| BA | SS | AC | BD

A -> a

B -> b

C -> SB

D -> SA

What should be done to construct a parse tree for the string

Exercise 6 -Solution

a a b b a b
0 1 2 3 4 5

	0	1	2	3	4	5
0	A	-	-	S	D	S
1		A	S	C	S	C
2			B	-	-	-
3				B	S	C
4					A	S
5						B

- For generating parse trees, modify the parse table d as below
- Every entry (i,j), i < j, of the table is a triple (N, s, p) which means that N accepts the sub-string from index i to j via a production of the form p: N -> N1 N2 and N1 accepts the substring from index i to (i+s-1) and N2 accepts the substring from index (i+s) to j

Exercise 6 –Solution [Cont.]

S -> AB (p1) | BA (p2) | SS (p3) | AC (p4) | BD (p5)

A -> a (p6)

B -> b (p7)

C -> SB (p8)

D -> SA (p9)

	0	1	2	3	4	5
0	A	-	-	(S,1,p4)	(D,4,p9)	(S,4,p3)
1		A	(S,1,p1)	(C,2,p8)	(S,2,p3)	(C,4,p8)
2			B	-	-	-
3				B	(S,1,p2)	(C,2,p8)
4					A	(S,1,p1)
5						B

See next slide for an algorithm for generating one parse tree given a table of the above form

Exercise 6 –solution [Cont.]

Algorithm for generating one parse tree starting from a nonterminal N for a sub-string (i,j)

ParseTree(N, i, j)

- If i = j, if N is in parseTable(i,j) return Leaf(N, w(i,j)) else report parse error
- Otherwise, pick an entry (N, s, p) from parseTable(i,j)
- If no such entry exist report that the sub-string cannot be parsed and return
- Let p: N -> N1 N2
- leftChild= ParseTree(N1, i, i+s-1)
- rightChild = ParseTree(N2, i+s, j)
- Return Node(N, leftChild, rightChild)