שאלה 1

שאלה 1.1: דיוק ה-parsing ה-dummy parser עבור משפטים באורך לכל היותר-40 מילים:

Bracketing Recall = 0.00

Bracketing Precision = 0.00

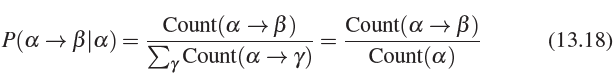
Bracketing FMeasure = -1.#J

שאלה 1.2: דיוק התיוג של ה-dummy parser עבור משפטים באורך לכל היותר 40 מילים:

Tagging accuracy = 18.78

שאלה 2

שאלה 2.1: הנוסחה לחישוב ההסתברויות לחוק (פרק 13 עמ' 8 בספר הלימוד):



שאלה 2.2: פסאודו קוד לבינרזיציה של עצים: (קוד רקורסיבי, הקריאה הראשונה מעבירה כפרמטר את שורש העץ)

**CFG\_TO\_CNF**(Input: ***node***)

1. If ***node*** has daughters:
   1. Invoke CFG\_TO\_CNF recursively on first (leftmost) daughter
   2. If there are more than two daughters:
      1. Create a new artificial node ***artificialNode*** (for node identifier scheme, see bellow)
      2. Remove redundant daughters from ***node*** and assign all “redundant” daughters to ***artificialNode***
   3. If (at this point) ***node*** has two daughters: (see explanation bellow)
      1. Invoke CFG\_TO\_CNF recursively on second (rightmost) daughter

הסבר לשלב 1.3: בשלב זה באלגוריתם, אחת משתי האפשרויות מתקיימת:

* ל-***node*** היו לכתחילה בת אחת או שתי בנות: במצב הזה, לא רץ שלב 1.2, ובשלב 1.3 המצב נותר כשהיה, ופשוט עובר על הבנים (במידה וקיימים) של הnode ושולח אותם כדי לבצע עליהם המרה ל-CNF.
* במידה ול-***node*** היו לכתחילה יותר משתי בנות: במצב הזה, שלב 1.2 רץ. בשלב 1.2 יוצרים בת "חדשה מלאכותית", וכל שאר הבנות (החל מהאינדקס השני, ועד לסוף רשימת הבנות), הופכים להיות בנות של הצומת המלאכותי. כלומר, שוב ל-***node*** יש שתי בנות - הבת השמאלית המקורית והבת הימנית המלאכותית החדשה.
  + במקרה הזה, נוספים ***n - 2*** בנות ל-***node*** (מספר הבנות הוא ***n***), שכן עבור כל בת (חוץ מהשתיים האחרונות), מוסיפים ***node*** ל-CYK matrix.

שאלה 2.2:

אלגוריתם הבינריזציה נדרש עבור הרצת אלגוריתם ה-CYK, שכן עבור CYK החוקים נדרשים להיות במבנה CNF. (במימוש שלנו, הוספנו לאלגוריתם תמיכה בחוקים אונאריים).

שאלה 2.3:

* כאשר מבצעים בינריזציה של העצים, כל צומת עם יותר מאח אחד מאבד מה"קונטקסט" שלו – הוא לא זוכר אילו אחים היו לו. כיוון שלקונטקסט יש חשיבות בשפה טבעית, איבוד מוחלט של הקונטקסט של כל צומת פוגע ביכולות הזיהוי הנכון והתיוג
* כאשר משתמשים ב-horizontal markovization, בעצם "מפצלים" חוק יחיד לכמה חוקים – חוק נפרד לכל שכנות של non terminals . למשל, אם נשתמש בדוגמה של מטלת הבית: לפני השינוי, ה-non-terminal X@// יכול היה להפיק- Y2, …, Yk . כעת, קיימים חוקים שונים (X@/Y2/, X@/Y3/, …), וכל חוק יכול להפיק תפוקות שונות. כלומר, הוספת הקונטקסט (קידוד האחים השמאליים), משנה את ה-parsing של המשפט.

**שאלה 3**

**Pre-processing**

1. Save map of lexical rules by the product word
2. Save map of unary grammar rules, by RHS argument
3. Save map of binary grammar rules, by leftmost RHS argument
4. Create the right upper-half of a matrix and populate it with empty maps (for holding different symbols probabilities, the matrix is indexed like Figure 13.4 (ch. 13, pp. 8 in the textbook)
5. Create a map for holding the backtrace (for reconstructing the probable tree)
6. Save ***default-prob*** the average probability of NN tag (for assigning to unknown words)

**PROBABILISTIC-CYK**(Input: ***sentence***)

*Part I: Initialization*

1. cyk-matrix ← new “half matrix”[[1]](#footnote-1) of length LEN(***sentence***), (if key doesn’t exist in map, default value to return is ∞ )
2. cyk-backtrace ← new empty map

*Part II: iterate over words in sentence*[[2]](#footnote-2)

1. For j ← 1 to LEN(***sentence***)

*Part II: Find lexical productions*

* 1. For each LHS of lexical rule {A | A 🡪 ***sentence***[j-1]}
     1. cyk-matrix[j-1, j, A] ← P(A 🡪 ***sentence***[j-1])
     2. cyk-backtrace[j-1, j, A] ← (-1, ***sentence***[j-1], null)[[3]](#footnote-3)

*Part III: iterate over unary rules after lexical productions*

* 1. Repeat till convergence or max iteration limit is reached:
     1. For each unary rule A 🡪 B which it’s RHS is in {B | (j-1,j,B) ∈ cyk-matrix}:[[4]](#footnote-4)
        1. If cyk-matrix[j-1,j,A] > PminusLogProb(A 🡪 B) + cyk-matrix[j-1,j,B]:
           1. cyk-matrix[j-1, j, A] ← PminusLogProb(A 🡪 B) + cyk-matrix[j-1,j,B]
           2. cyk-backtrace[j-1, j, A] ← (-2, A null)[[5]](#footnote-5)
  2. for i ← j – 2 down to 0 do:

*Part IV: iterate over binary rules, search for each possible split k between i and j*

* + 1. for k ← i + 1 to j-1 do:
       1. for each binary rule “A 🡪 B C” where cyk-matrix[i,k,B] < ∞ AND cyk-matrix[k,j,C] < ∞ :4
          1. if cyk-matrix[i,j,A] > PminusLogProb(A 🡪 B C) + cyk-matrix[i,k,B] + cyk-matrix[k,j,C]: // (i.e. the new rule production is more probable for A)

cyk-matrix[i,k,A] ← PminusLogProb (A 🡪 B C) + cyk-matrix[i,k,B] + cyk-matrix[k,j,C]

cyk-backtrace[i,j,A] ← (k,B,C)

*Part V: iterate over all unary rules in cell [i,j]*

* + 1. Repeat till convergence or max iteration limit is reached:
       1. For each unary rule A 🡪 B which it’s RHS is in {B | (j-1,j,B) ∈ cyk-matrix}:4
          1. If cyk-matrix[j-1,j,A] > PminusLogProb(A 🡪 B) + cyk-matrix[j-1,j,B]:

cyk-matrix[j-1, j, A] ← PminusLogProb(A 🡪 B) + cyk-matrix[j-1,j,B]

cyk-backtrace[j-1, j, A] ← (-2, A null)5

1. return BUILD-TREE(cyk-matrix, cyk-backtrace, start-symbols)

ניתוח סיבוכיות זמן ריצה כפונקציה של אורך המשפט N, ו-G גודל הדקדוק:

הלולאה החיצונית j רצה N איטרציות. הלולאות הפנימיות i,k רצות מספר משתנה של פעמים, כאשר מספר האיטרציות בכל לולאה חסום מלמעלה בערך משתני הלולאות החיצוניות- i,j . המספרים הללו חסומים ב- O(n3). באלגוריתם CYK המקורי עוברים מספר קבוע של פעמים על כל החוקים בדקדוק, כלומר, סיבוכיות האלגוריתם היא- O(n3∙G) . זהו גם זמן הריצה במימוש שלנו, במקרה הגרוע. בפועל, יצרנו אינדקס כך שבפועל לא היה מעבר על כל החוקים בכל איטרציה, מה ששיפר משמעותית את זמני הריצה.

ניתוח סיבוכיות המקום כפונקציה של אורך המשפט N, ו-G גודל הדקדוק:

שאלה 4:

פסאודו קוד עבור השחזור:

עבור h=0:

Bracketing Precision = 44.47

Bracketing FMeasure = 33.14

Complete match = 0.00

Average crossing = 3.98

No crossing = 21.33

2 or less crossing = 43.78

Tagging accuracy = 79.54

עבור h=1:

\_\_\_\_\_\_\_\_\_\_\_\_ len<=40 \_\_\_\_\_\_\_\_\_\_\_\_\_

Number of sentence = 900

Bracketing Recall = 26.32

Bracketing Precision = 43.73

Bracketing FMeasure = 32.86

Complete match = 0.00

Average crossing = 4.04

No crossing = 21.89

2 or less crossing = 43.67

Tagging accuracy = 77.89

עבור h=2:

\_\_\_\_\_\_\_\_\_\_\_\_ len<=40 \_\_\_\_\_\_\_\_\_\_\_\_\_

Number of sentence = 1350

Bracketing Recall = 26.69

Bracketing Precision = 43.19

Bracketing FMeasure = 32.99

Complete match = 0.00

Average crossing = 4.19

No crossing = 21.48

2 or less crossing = 42.15

Tagging accuracy = 77.21

עבור h=-1:

\_\_\_\_\_\_\_\_\_\_\_\_ len<=40 \_\_\_\_\_\_\_\_\_\_\_\_\_

Number of sentence = 1800

Bracketing Recall = 26.90

Bracketing Precision = 42.84

Bracketing FMeasure = 33.05

Complete match = 0.00

Average crossing = 4.28

No crossing = 21.33

2 or less crossing = 41.67

Tagging accuracy = 76.88

הסתכלנו על ה10 המשפטים הראשונים עבור h=0 (התוצאה הטובה ביותר):

סווג טעויות מהעלים של העץ לכיוון השורש:

· יש מספר מילים(סגמנטים)

שאלה 5:

השתמשנו באולגוריתם parent encoding - מוסיפים את האבא לכל אחד מהבנים, ככה שהחוקים מודיעים גם למי האבא, ואז זה ממש דומה לbi-gram(כמו שעשינו בממן 12).

Sentence #10: **IFRAL ARD yyCM SMNKUL H BIJWX H LAWMI yyCM AMR KI MMLA MQWM FR H EBWDH W H RWWXH yyCM DWD MGN yyCM HQIM WEDH BIN yyDASH MFRDIT yyCM F HMLICH LHGDIL B AWPN MFMEWTI AT H QNSWT L H MESIQIM yyDOT**

משפט: **ישראל ארד, סמנכ"ל הביטוח הלאומי, אמר כי ממלא מקום שר העבודה והרווחה, דוד מגן, הקים ועדה בין-משרדית, שהמליצה להגדיל באופן משמעותי את הקנסות למעסיקים.**

השוואה בין הניתוח הנכון של המשפט ("gold parsing"), לבין הניתוח של האלגוריתם: **שגיאה! אובייקט מוטבע לא חוקי.**

1. ראה Figure 13.4 בפרק 13 בספר הקורס (עמ' 8). האינדקסים באלגוריתם מתייחסים גם הם לאינדקסים של התאים בטבלה. [↑](#footnote-ref-1)
2. זה לא קטע קוד נפרד (שאר הקטעים מקוננים בלולאה הזו), אבל מטעמי בהירות אנו מתייחסים ללוגיקה / משמעות של הלולאה הזו בנפרד. [↑](#footnote-ref-2)
3. המספר -1 מסמן את ה-triplet כחוק כ-triplet טרמינלי [↑](#footnote-ref-3)
4. חיפוש משופר בזכות אינדוקס החוקים לפי הארגומנט השמאלי של ה-RHS שלהם (חוקים אונריים ובינאריים בנפרד). [↑](#footnote-ref-4)
5. המספר- -2 מסמן את ה-triplet כתוצר של חוק אונרי, ולפיכך הערכים בו מתפרשים באופן קצת שונה. [↑](#footnote-ref-5)