yz709-FML-sup1

Natural languages

Question 1

1. The following natural language sentences are ambiguous. Describe the ambiguities.
   (a) *She fed her cat food.*
   (b) *She saw the man with one eye.*
   (c) *She saw the queen in the garden with the telescope.*

- (a): We can interpret cat food as a single noun, or two nouns
  - She[pron] fed[verb] her[pron] cat food[noun]: A gives cat food to B
  - She[pron] fed[verb] her cat [noun] food [noun]: A gives some food to her own cat.
- (b): the phrase `with one eye` can be used to describe she or the man
  - Describes she: she used one eye and saw the man
  - Describes the man: she saw that there is a man with one eye

- (c): both the phrase `in the garden` and the phrase `with the telescope` can be used to describe `she` or the `queen`.
    - She used a telescope and saw the queen who is the in garden.
    - She was in the garden and used the telescope to see the queen.
    - The queen is in the garden with the telescope, and she saw the queen.

> 💡 Comments:

Question 2

2. The following natural language sentences are difficult to process. Hypothesise what is causing the difficulty.
    (a) *I told the girl the rabbit knew the caterpillar would help her.*
    (b) *The twins the rabbit the girl chased liked laughed.*
    (c) *She shook the bottle containing the potion which had made her grow very tall up.*

    Discuss how we might test your hypotheses.

- Hypothesis:
    - (1) Each sentence has a derivation tree, if this tree has a small probability of occurring (i.e., the tree is surprising), then we need more cognitive effort in parsing this sentence;
    - (2) When parsing the sentence using a pushdown automaton, the maximum depth of the stack is higher if the sentence requires more memory load, thus more difficult to parse.
- Testing:
    - (1) Generate the derivation tree for a sentence, assign probabilities to all possible partial derivations and multiply to get the total probability for a particular derivation. Compute the correlation between the probability of a particular derivation tree and the difficulty of parsing for a sentence.
    - (2) Given a large sentence base, compute the maximum depth of the stack for each sentence, at the same time, each sentence is labelled its difficulty of parsing, compute the correlation between the maximum stack depth and the difficulty of parsing for a sentence.

💡 Comments:

Formal languages

Properties of regular and context free languages

Question 1

1. (reminder question) If $\mathcal{L}_1$ and $\mathcal{L}_2$ are regular languages prove the following are also regular:
   (a) $\mathcal{L}_1 \cup \mathcal{L}_2$
   (b) $\mathcal{L}_1 \mathcal{L}_2$
   (c) $\mathcal{L}_1 \cap \mathcal{L}_2$

- (a):
  - Let R_1 and R_2 be the regular expressions for regular languages $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. Then the regular expression $R = R_1 + R_2$ acceptes $\mathcal{L}_1 \cup \mathcal{L}_2$ by the definition of regular expressions, i.e., $\mathcal{L}(R) = \mathcal{L}(R_1 + R_2) = \mathcal{L}(R_1) \cup \mathcal{L}(R_2)$, since the regular expression R accepts $\mathcal{L}_1 \cup \mathcal{L}_2$, the union is regular.
- (b):
  - Let R_1 and R_2 be the regular expressions for regular languages $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. Then the regular expression $R = R_1.R_2$ acceptes $\mathcal{L}_1\mathcal{L}_2$ by the definition of regular expressions, i.e., $\mathcal{L}(R) = \mathcal{L}(R_1.R_2) = \mathcal{L}(R_1).\mathcal{L}(R_2)$, since the regular expression R accepts $\mathcal{L}_1.\mathcal{L}_2$, the concatenation is regular.
- (c):
  - $\mathcal{L}_1 \cap \mathcal{L}_2 = \overline{\overline{\mathcal{L}_1} \cup \overline{\mathcal{L}_2}}$
  - Since regular languages are closed under complementation and union, we can prove regular languages are closed under intersection.
    - We can prove regular languages are closed under complementation by assuming we have a DFA M that accepts $\mathcal{L}$, if we swap the accepting

states and non-accepting states, then this DFA accepts the language $\overline{\mathcal{L}}$, hence it is a regular language.

- If $\mathcal{L}_1$ and $\mathcal{L}_2$ are regular languages, then $\overline{\mathcal{L}_1}$ and $\overline{\mathcal{L}_2}$ are gular languages, $\overline{\mathcal{L}_1} \cup \overline{\mathcal{L}_2}$ is a regular language and $\overline{\overline{\mathcal{L}_1} \cup \overline{\mathcal{L}_2}}$ is a regular language.

💡 Comments:

Question 2

2. If $\mathcal{L}_1$ is regular and $\mathcal{L}_2$ is context free prove the following is also context free:
   (a) $\mathcal{L}_1 \cap \mathcal{L}_2$

- Since $\mathcal{L}_1$ is a regular language, it can be recognised by a DFA $M = (\mathcal{Q}_1, \Sigma, \Delta_1, s_1, \mathcal{F}_1)$ and $\mathcal{L}_2$ is a context-free language, it can be recognised by a PDA $P = (\mathcal{Q}_2, \Sigma, \Gamma, \Delta_2, s_2, \perp, \mathcal{F}_2)$.
- Define a new PDA $P_\cap = (\mathcal{Q}, \Sigma, \Gamma, \Delta, s, \perp, \mathcal{F})$ where $\mathcal{Q} = \mathcal{Q}_1 \times \mathcal{Q}_2, s = (s1, s2), \mathcal{F} = \mathcal{F}_1 \times \mathcal{F}_2$
- The transition function $\Delta \subseteq ((\mathcal{Q}_1 \times \mathcal{Q}_2) \times (\Sigma \cup \epsilon) \times \Gamma) \times ((\mathcal{Q}_1 \times \mathcal{Q}_2) \times \Gamma^*)$, written as $(q_1', \beta) = \delta_1(q_1, a, X)$ and $q_2' = \delta_2(q_2, a)$
  - When the PDA P is in state $q_1 \in \mathcal{Q}_1$, reading $a \in (\Sigma \cup \{\varepsilon\})$ with $X \in \Gamma$ on top of stack, it can move to state $q_1' \in \mathcal{Q}_1$ and replace X with $\beta \in \Gamma$.
  - **And** at the same time the DFA M is in state $q_2 \in \mathcal{Q}_2$, reading the same symbol $a \in (\Sigma \cup \{\varepsilon\})$, will transfer to state $q_2' \in \mathcal{Q}_2$.
- Hence for any input strings, the new PDA $P_\cap$ accepts that string if it can be accepted by both the DFA M and the PDA P, hence $\mathcal{L}(P_\cap) = \mathcal{L}_1 \cap \mathcal{L}_2$

💡 Comments:

Pumping lemma for regular and context free languages

Question 1

1. (reminder question) Use the pumping lemma for regular languages to prove that the following are not regular:
   (a) $\mathcal{L} = \{ab^n cd^n e | n \geq 1\}$
   (b) $\mathcal{L} = \{a^n b^{n+1} | n \geq 1\}$
   (c) $\mathcal{L} = \{ww | w \in \{a, b\}^*\}$

- (a):
  - For $L \geq 1$, consider $w = ab^L cd^L e \in \mathcal{L}$ (hence w is of length $\geq$ the number of states in DFA that accepts $\mathcal{L}$), we split $w = u_1 v u_2$ such that $u_1 = ab^r$, $v = b^s$, $u_2 = b^{L-r-s} cd^L e$, where $s > 0, r + s + 1 \leq L$
  - According to the pumping lemma property of a regular language, for all $n \geq 0, u_1 v^n u_2 \in \mathcal{L}$, that is, we can go into the loop many times inside the DFA. However, when $n = 0$, we have $u_1 v^0 u_2 = ab^r (b^s)^0 b^{L-r-s} cd^L e = ab^{L-s} cd^L e \notin \mathcal{L}$ because $L - s < L$, thus $\mathcal{L}$ is not a regular language.
- (b):
  - For $L \geq 1$, consider $w = a^L b^{L+1} \in \mathcal{L}$ (hence w is of length $\geq$ the number of states in DFA that accepts $\mathcal{L}$), we split $w = u_1 v u_2$ such that $u_1 = a^r$, $v = a^s$, $u_2 = a^{L-r-s} b^{L+1}$, where $s > 0, r + s \leq L$
  - According to the pumping lemma property of a regular language, for all $n \geq 0, u_1 v^n u_2 \in \mathcal{L}$, that is, we can go into the loop many times inside the DFA. However, when $n = 0$, we have $u_1 v^0 u_2 = a^r (a^s)^0 a^{L-r-s} b^{L+1} = a^{L-s} b^{L+1} \notin \mathcal{L}$ because $L - s < L$, thus $\mathcal{L}$ is not a regular language.
- (c):
  - For $L \geq 1$, consider $w = a^L b^L a^L b^L \in \mathcal{L}$ (hence w is of length $\geq$ the number of states in DFA that accepts $\mathcal{L}$), we split $w = u_1 v u_2$ such that $u_1 = a^r$, $v = a^s$, $u_2 = a^{L-r-s} b^L a^L b^L$, where $s > 0, r + s \leq L$
  - According to the pumping lemma property of a regular language, for all $n \geq 0, u_1 v^n u_2 \in \mathcal{L}$, that is, we can go into the loop many times inside the DFA. However, when $n = 0$, we have $u_1 v^0 u_2 = a^r (a^s)^0 a^{L-r-s} b^L a^L b^L = a^{L-s} b^L a^L b^L \notin \mathcal{L}$ because $L - s < L$, thus $\mathcal{L}$ is not a regular language.

💡 Comments:

Question 2

2. Use the pumping lemma for context free languages to prove that the following are not context free:
   (a) $\mathcal{L} = \{a^n b^n c^n | n \geq 1\}$
   (b) $\mathcal{L} = \{a^n b^n c^m | n \leq m\}$

- (a):
  - For $L \geq 1$, consider $w = a^L b^L c^L \in \mathcal{L}$ (hence w is of length $\geq$ the number of states in DFA that accepts $\mathcal{L}$), we split $w = u_1 y u_2 z u_3$ such that $u_1 = a^r$, $y = a^s$, $u_2 = a^t$, $z = a^p$, $u_3 = a^{L-r-s-t-p} b^L c^L$ where $s + p > 0, s + t + p \leq L$
  - According to the pumping lemma property of a context-free language, for all $n \geq 0, u_1 y^n u_2 z^n u_3 \in \mathcal{L}$, that is, we can pop an item n times into the stack and pop it off from the stack n times. However, when $n = 0$, we have $u_1 y^0 u_2 z^0 u_3 = a^r (a^s)^0 (a^t)(a^p)^0 a^{L-r-s-t-p} b^L c^L = a^{L-s-p} b^L c^L \not\in \mathcal{L}$ because $L - s - p < L$, thus $\mathcal{L}$ is not a context free language.
  - Intuitively, there is no way we can split w into five parts satisfying the requirements that both a and c will be in yu_2z, hence if $a \in yz$, then $|u_1u_2u_3|_{\#a} < |u_1u_2u_3|_{\#c}$, if $c \in yz$, then $|u_1u_2u_3|_{\#a} > |u_1u_2u_3|_{\#c}$ and if neither are in yz, then $|u_1u_2u_3|_{\#b} < |u_1u_2u_3|_{\#a} = |u_1u_2u_3|_{\#c}$.
- (b):
  - For $L \geq 1$, consider $w = a^L b^L c^L \in \mathcal{L}$ (hence w is of length $\geq$ the number of states in DFA that accepts $\mathcal{L}$), we split $w = u_1 y u_2 z u_3$ such that $u_1 = a^L b^L c^{L-s-t-p-r}$, $y = c^s$, $u_2 = c^t$, $z = c^p$, $u_3 = c^r$ where $s + p > 0, s + t + p \leq L$
  - According to the pumping lemma property of a context-free language, for all $n \geq 0, u_1 y^n u_2 z^n u_3 \in \mathcal{L}$, that is, we can pop an item n times into the stack and pop it off from the stack n times. However, when $n = 0$, we have $u_1 y^0 u_2 z^0 u_3 = a^L b^L c^{L-s-t-p-r} (c^s)^0 c^t (c^p)^0 c^r = a^L b^L c^{L-s-p} \not\in \mathcal{L}$ because $L - s - p < L$, thus $\mathcal{L}$ is not a context free language.

💡 Comments:

Top-down parsing of context free grammars

Question 1

1. Write an implementation of the Earley parser that can use the toy grammar from Lecture 3 to parse the sentences below:
   (a) They can fish in rivers.
   (b) They can fish in rivers in December.

   How many parses are there for each sentence?

Don't over-engineer this, you just need to implement the algorithm to build the chart—you don't need write code to print derivation trees etc. The point of this exercise is to help you think through the algorithm. Your supervisor doesn't need to see the code, this is not a tick.

```
(* toy grammar as a reference *)
N = {S,NP,VP,PP,N,V,P}
Σ = {can,fish,in,rivers,they,December,...}
S = S
P = {S -> NP VP
     NP -> N PP | N
     PP -> P NP
     VP -> VP PP | V VP | V NP | V
     P -> in | ...
     V -> can | fish | ...}

(a)
(* 0 they 1 can 2 fish 3 in 4 rivers 5 *)
ID      RULE               [Start,End]  HIST
e0     S -> .NP VP         [0,0]                  (induction step)   word 0
e1     NP -> .N PP         [0,0]                  (predict step)     word 1
e2     NP -> .N            [0,0]                  (predict step)
e3     N -> they.          [0,1]                  (scan step)
e4     NP -> N.            [0,1]         e3       (complete step)
e5     NP -> N. PP         [0,1]         e3       (complete step)
e6     S -> NP.VP          [0,1]         e4       (complete step)
e7     PP -> .P NP         [1,1]                  (predict step)     word 2
e8     VP -> .V            [1,1]                  (predict step)
e9     VP -> .V NP         [1,1]                  (predict step)
e10    VP -> .V VP         [1,1]                  (predict step)
e11    VP -> .VP PP        [1,1]                  (predict step)
e12    V -> can.           [1,2]                  (scan step)
e13    VP -> V.            [1,2]         e12      (complete step)
e14    VP -> V.NP          [1,2]         e12      (complete step)
e15    VP -> V.VP          [1,2]         e12      (complete step)
e16    S -> NPVP.          [0,2]        e4,e13    (complete step)
e17    VP -> VP.PP         [1,2]         e13      (complete step)
e18    NP -> .N            [2,2]                  (predict step)     word 3
e19    NP -> .N PP         [2,2]                  (predict step)
e20    VP -> .V            [2,2]                  (predict step)
e21    VP -> .V NP         [2,2]                  (predict step)
e22    VP -> .V VP         [2,2]                  (predict step)
e23    VP -> .VP PP        [2,2]                  (predict step)
e24    PP -> .P NP         [2,2]                  (predict step)
e25    V -> fish.          [2,3]                  (scan step)
e26    VP -> V.            [2,3]         e25      (complete step)
```

```
e27     VP -> V.NP          [2,3]         e25      (complete step)
e28     VP -> V.VP          [2,3]         e25      (complete step)
e29     S -> NPVP.          [0,3]       e4,e13,e26 (complete step)
e30     VP -> VP.PP         [2,3]         e26      (complete step)
e31     NP -> .N            [3,3]                  (predict step)     word 4
e32     NP -> .N PP         [3,3]                  (predict step)
e33     VP -> .V            [3,3]                  (predict step)
e34     VP -> .V NP         [3,3]                  (predict step)
e35     VP -> .V VP         [3,3]                  (predict step)
e36     VP -> .VP PP        [3,3]                  (predict step)
e37     PP -> .P NP         [3,3]                  (predict step)
e38     P -> in.            [3,4]                  (scan step)
e39     PP -> P.NP          [3,4]         e38      (complete step)
e40     NP -> .N            [4,4]                  (predict step)     word 5
e41     NP -> .N PP         [4,4]                  (predict step)
e42     N -> rivers.        [4,5]                  (scan step)
e43     NP -> N.            [4,5]         e42      (complete step)
e44     NP -> N.PP          [4,5]         e42      (complete step)
e45     PP -> P NP.         [3,5]        e38,e43   (complete step)
e46     VP -> VP PP.        [2,5]      e26,e38,e43 (complete step)
e47     S -> NP VP.         [0,5]e4,e13,e26,e38,e43(complete step)

(b)
(* 0 they 1 can 2 fish 3 in 4 rivers 5 in 6 December 7 *)
continue ...
e48     PP -> .P NP       [5,5]                    (predict step)     word 6
e49     P -> in.          [5,6]                    (scan step)
e50     PP -> P.NP        [5,6]           e46      (complete step)
e51     NP -> .N          [6,6]                    (predict step)     word 7
e52     NP -> .N PP       [6,6]                    (predict step)
e53     N -> December.    [6,7]                    (scan step)
e54     NP -> N.          [6,7]           e50      (complete step)
e55     NP -> N.PP        [6,7]           e50      (complete step)
e56     PP -> P NP.       [5,7]          e46,e51   (complete step)
e57     NP -> N PP.       [4,7]        e42,e46,e51 (complete step)
e58     PP -> P NP.       [3,7]     e38,e42,e46,e51(complete step)
e59     VP -> VP PP.      [2,7]  e26,e38,e42,e46,e51(complete step)
e60     S -> NP VP.       [0,7]e4,e13,e26,e38,e42,e46,e51(complete step)
```

💡 Comments:
In the predict step at ID `e1`, we have expanded by adding the dotted rule `NP -> . N PP`, but this rule can be further expanded to `N -> .can | .fish | ...`, why that doesn't need to be added as an expanded dotted rule in the predict step? Similarly when we added the dotted rule `PP -> .P NP` at ID `e7`, why we don't further expand and add the dotted rule `P -> .in | ....`?

Comparing grammar formalisms

Question 1

1. Consider the following sentences:
   - *Alice eats cakes.*
   - *The caterpillar gives Alice cakes.*
   - *The cat with a grin disappears.*
   - *Alice paints white roses red.*

   Using the examples in the notes/slides to start you off, complete the following tasks:

   (a) Define a context free grammar that could generate the sentences.

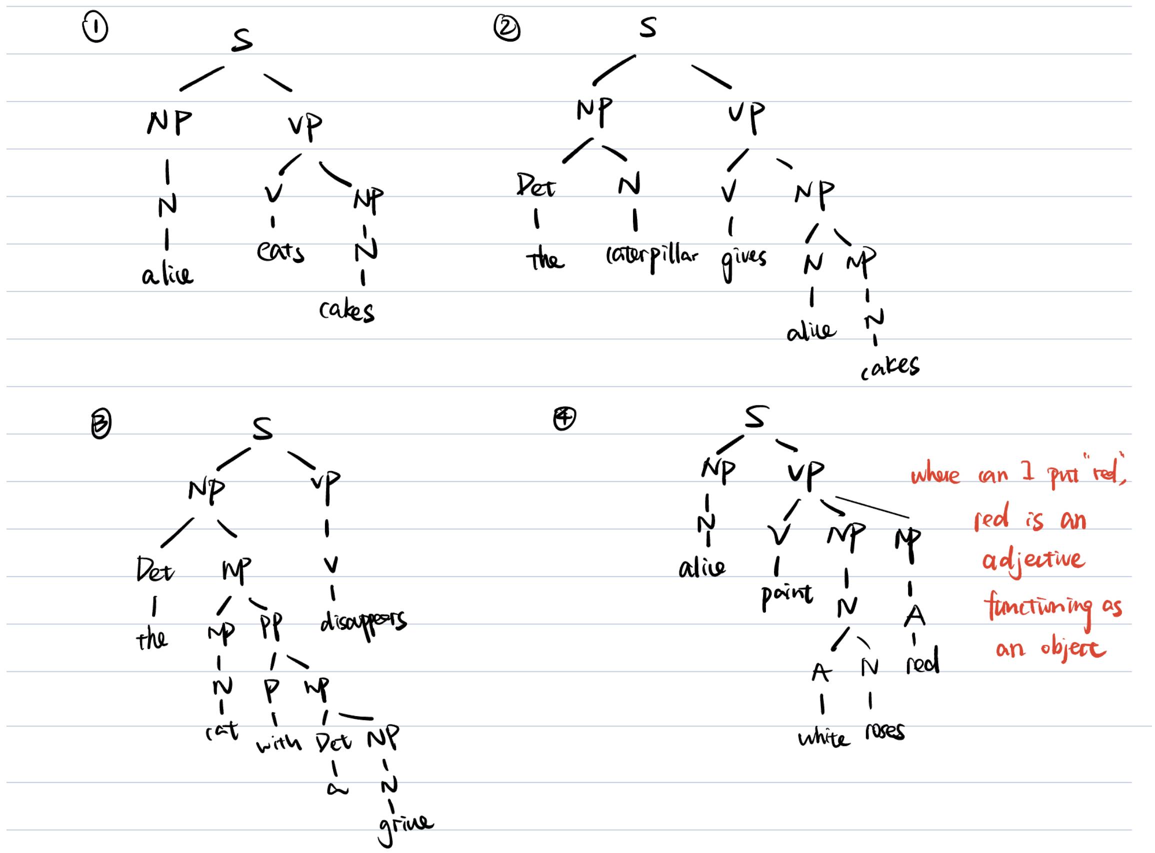
   (b) Draw a dependency parse for the sentences.

   (c) Define a tree adjoining grammar that could generate the sentences.

> A grammar structure containing `([determiner, noun] [adjective, verb] [preposition, determiner, noun])`

- (a):

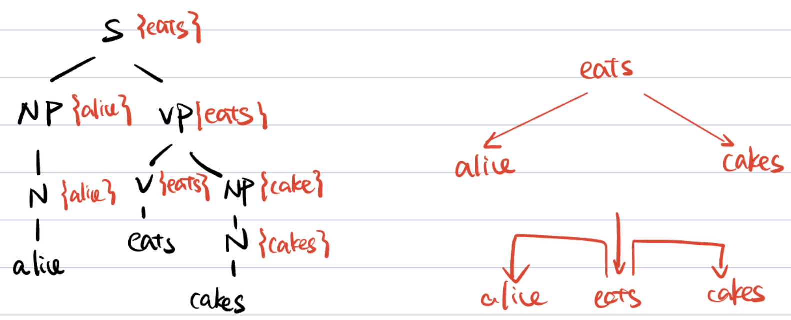
```
S -> NP VP
NP -> N | Det NP | N NP | NP PP | A
Det -> a | the
N -> A N|alice|cakes|caterpillar|cat|grin|roses
VP -> VP PP | V NP | V | V NP PP | V PP
V -> eats|gives|disappears|paints
PP -> P NP
P -> with
A -> red|white
```



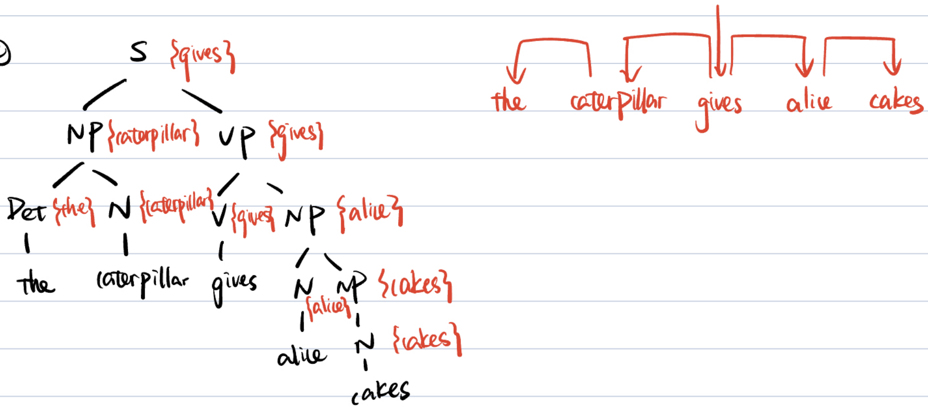
- (b):

```
Σ = {alice,cakes,caterpillar,cat,grin,roses,
      eats,gives,disappears,paints,red,white,a,the,with}
D = {L,R}
s = (1) alice; (2) the; (3) the; (4) alice
P = {(eats -> alice,L | cakes, R)
     (caterpillar -> the, L)
     (gives -> caterpillar, L | alice, R)
     (alice -> cakes, R)
     (cat -> the, L | with, R)
     (with -> grine, R)
     (grine -> a, L)
     (disappears -> cat, L)
     (paint -> alice, L | red, R | roses, R)
     (roses -> white, L)}
```

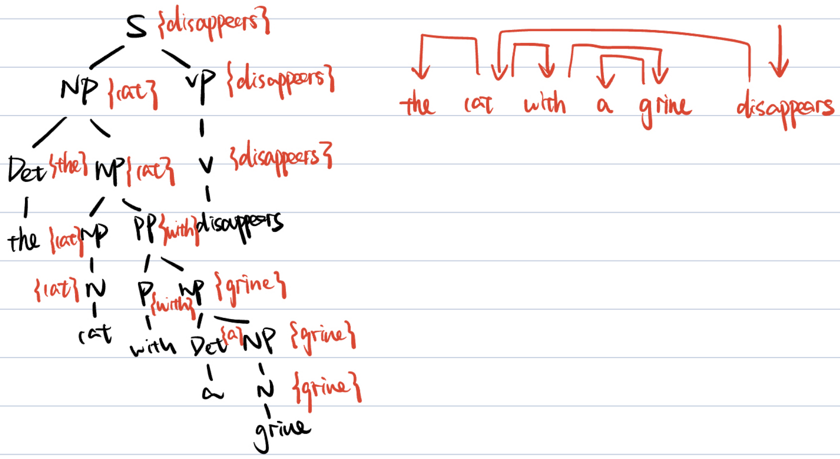
①



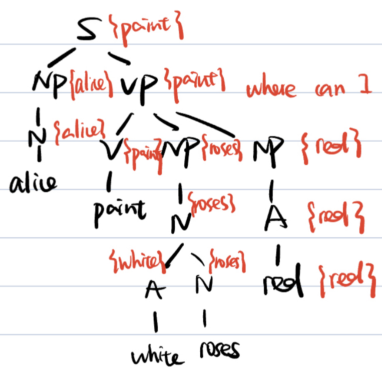
②



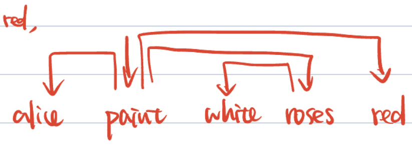
③



④

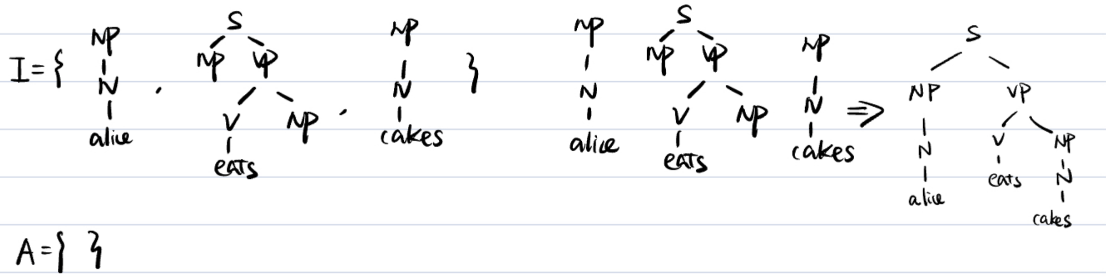


where can I put "red",

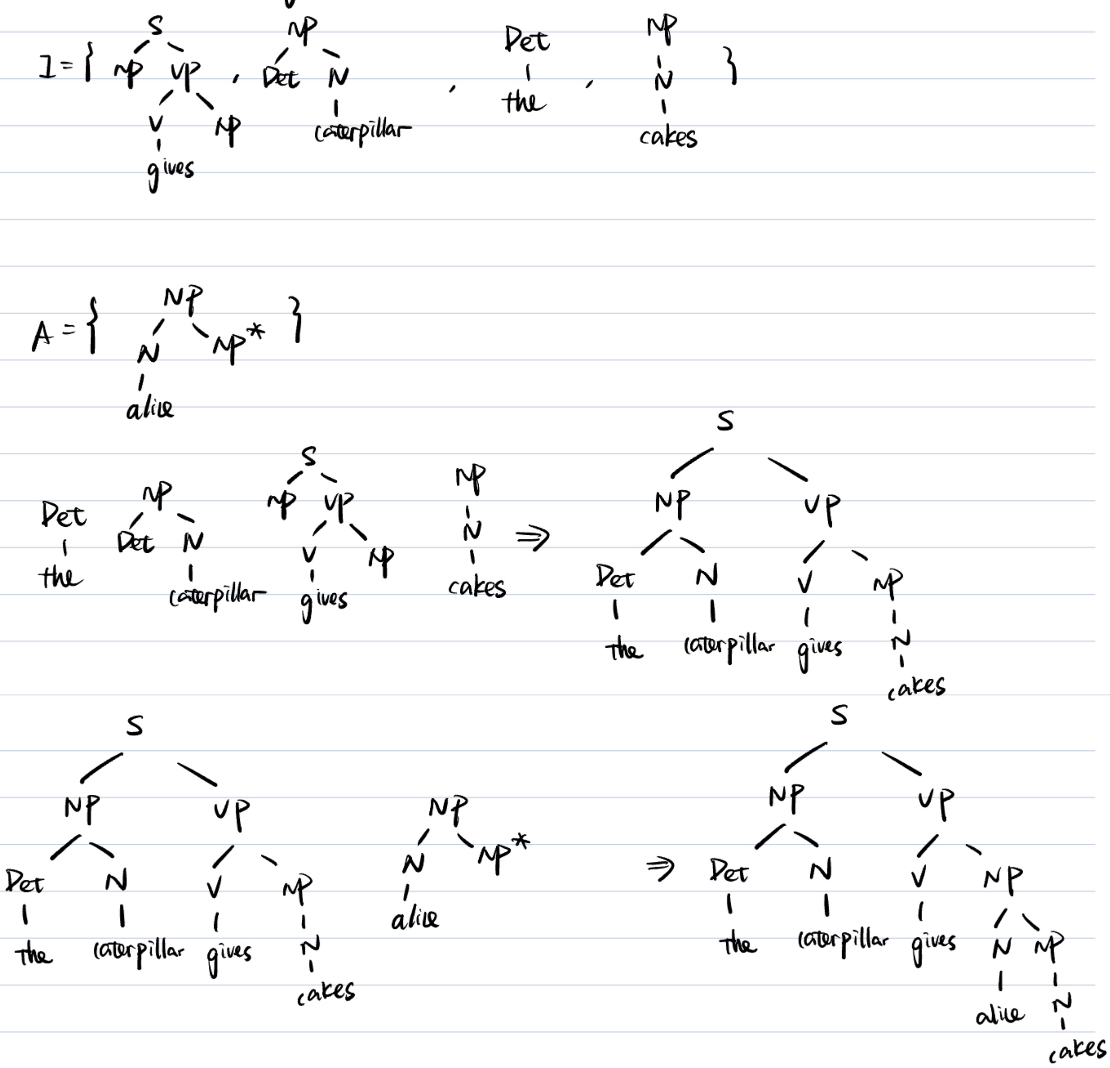


- (c):

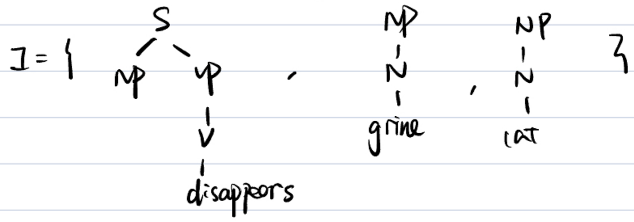
① alice eats cakes

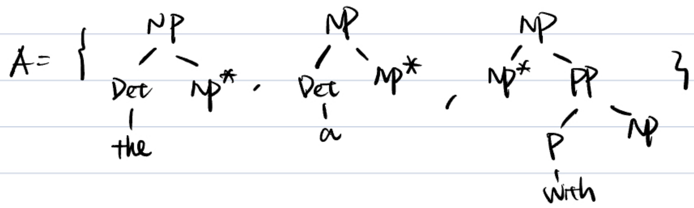


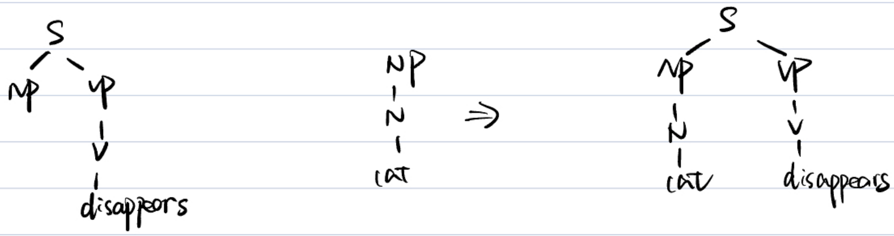
② the caterpillar gives alice cakes

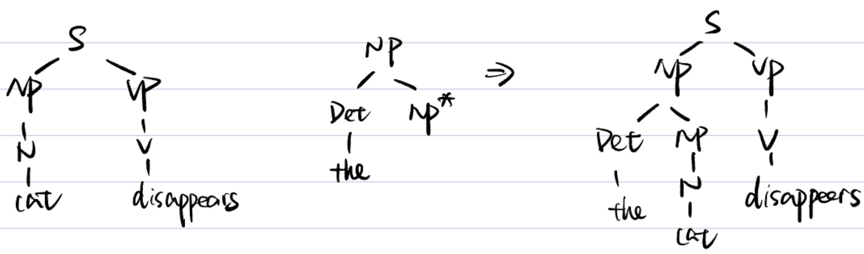


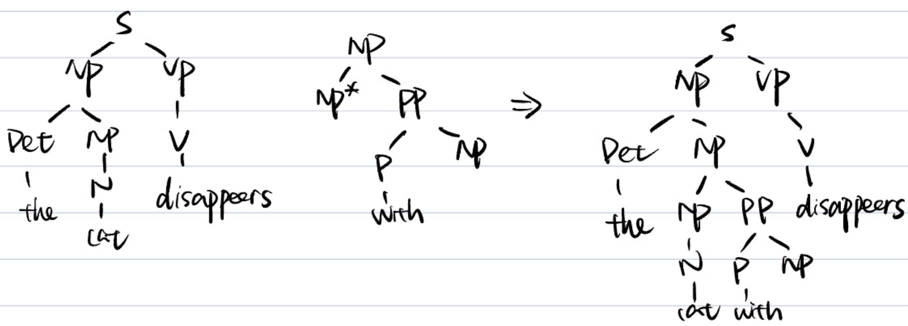
③ the cat with a grine disappers

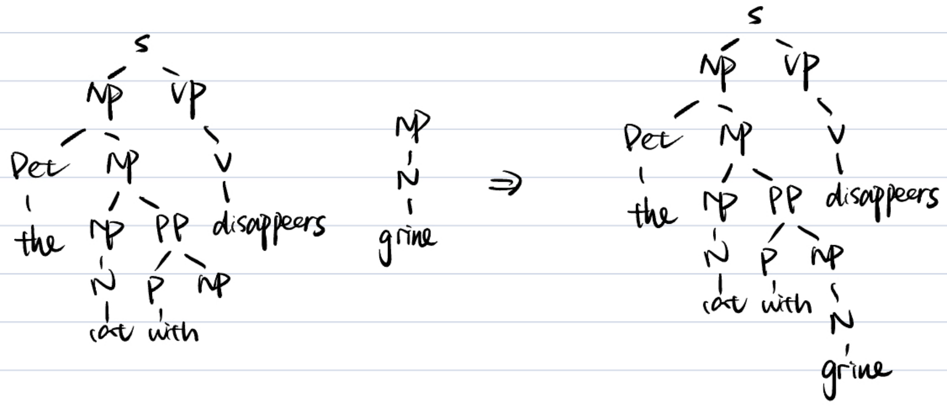


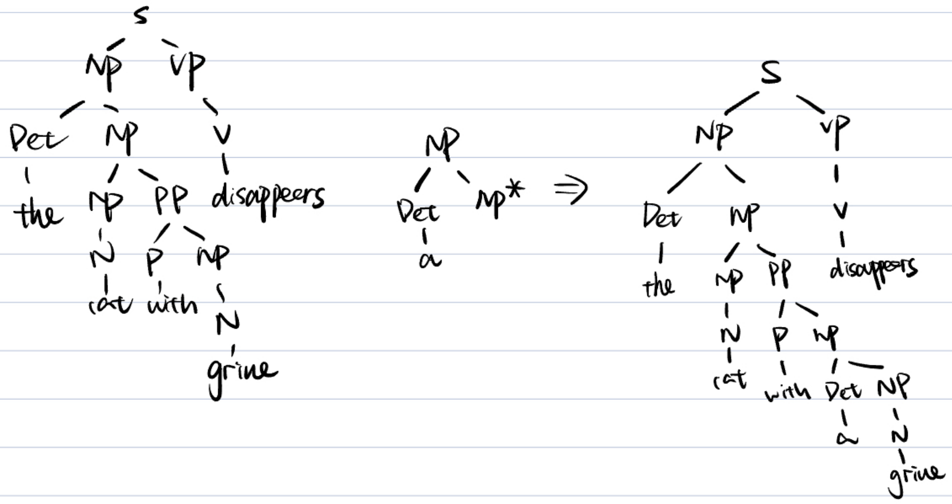




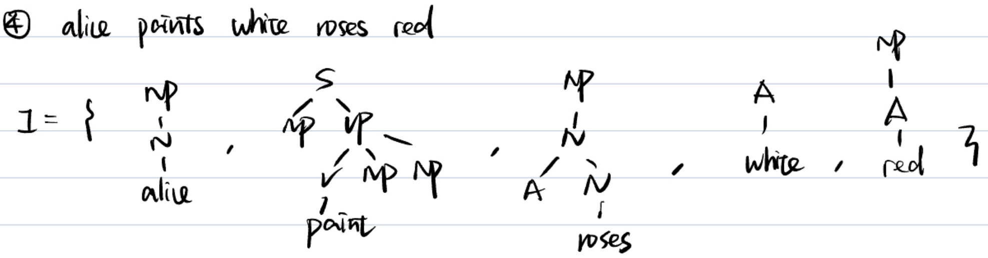




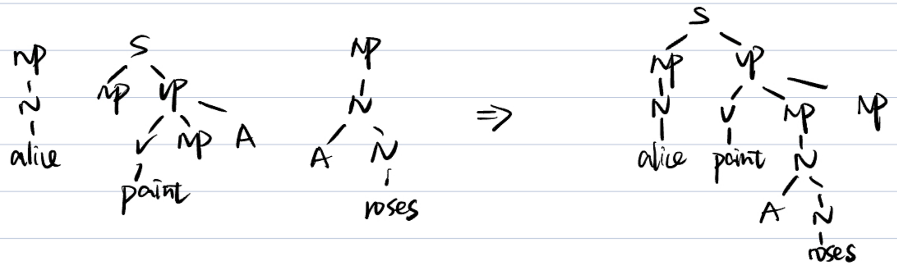

S
NP VP
Det NP V
the NP PP disappeers
N P NP
cat with
NP
N
grine
⇒
S
NP VP
Det NP V
the NP PP disappeers
N P NP
cat with N
grine

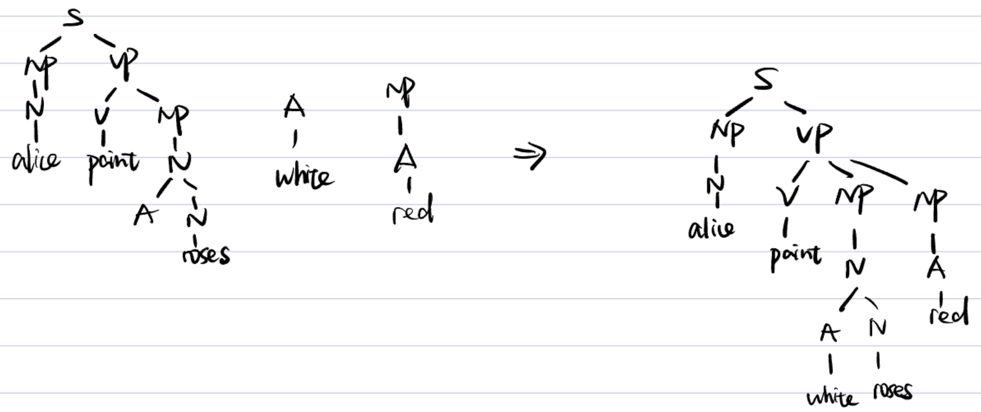

S
NP VP
Det NP V
the NP PP disappeers
N P NP
cat with N
grine
NP
Det NP*
a
⇒
S
NP VP
Det NP V
the NP PP disappeers
N P NP
cat with Det NP
a N
grine

④ alice points white roses red



A = { }





💡 Comments:

Questions

1. When we prove English is not a regular language, we used the property that a regular language is closed under intersection, $\mathcal{L}_{eng} \cap \mathcal{L}_{a^*b^*} = \mathcal{L}_{a^nb^n}$, I don't understand why English intersects with a^*b^* would produce a^nb^n.