HUGH HAN, XIAOCHEN LI, SHIJIE WU

600.465 Natural Language Processing **Fall 2016**
Homework #1 **Due:** 21 September 2016, 2pm

1 Random Sentence Generator

The following output was produced using `./randsent grammar 10`.

```
is it true that every sandwich under every floor kissed the sandwich ?
is it true that the pickle under the pickle on a delicious pickled president pickled the
    president under the pickle ?
the president on the president in every fine floor under the sandwich in every president
    under a chief of staff ate the floor under a chief of staff on the pickle on a floor in
    every chief of staff with the perplexed president on the president .
a chief of staff pickled the sandwich in a floor on the sandwich with the president with the
    fine floor in every sandwich in a floor on a pickle on every sandwich with every sandwich
    under the pickle under a sandwich on the president under a pickled sandwich in the
    delicious floor on the pickled pickled pickle under the chief of staff in every president
    on the perplexed pickle on the pickle in every president on every floor on a sandwich in
    the chief of staff in every pickle in the pickle with a floor on the sandwich with every
    floor on a president with a chief of staff with a floor on every floor in every chief of
    staff in a sandwich on every floor on the president on the president under the chief of
    staff in every pickle under every chief of staff with every pickle with every pickle on
    every president on the pickle .
a pickle on the sandwich with every sandwich on the chief of staff under the chief of staff
    on a chief of staff under every pickle with every president in a pickle under every pickle
    on a chief of staff with the pickle under every pickle on a chief of staff under every
    delicious sandwich under a pickle with every president under the president in the sandwich
    in every pickle in the sandwich under the pickled chief of staff with every sandwich under
    the floor with a delicious floor with a pickle in a pickled chief of staff under every
    floor under every sandwich in every floor under every chief of staff under every chief of
    staff in every sandwich with the chief of staff with the president under the pickle under
    every pickle with every pickle in the floor in every president with the president with a
    pickled sandwich on a president under a floor in the fine floor in a chief of staff under
    a chief of staff under every floor in a perplexed fine floor with a president with the
    president under a president in the delicious delicious floor on every pickle on the pickle
    under a fine pickle with the chief of staff with every chief of staff on the sandwich in
    every sandwich on every chief of staff in the sandwich under a pickle in every delicious
    perplexed sandwich with the floor with a perplexed chief of staff with the president on
    the floor on the pickle in a pickle under every floor ate the chief of staff !
every delicious president wanted the fine pickle .
is it true that the pickle wanted the sandwich ?
a floor wanted a president !
every pickle with every chief of staff kissed a delicious fine president with a president .
a chief of staff with a floor pickled the perplexed perplexed fine sandwich .
```

2 Questions

(a) The rule

$$\texttt{NP} \rightarrow \texttt{NP PP}$$

is responsible for all these long sentences.

To explain why, let's first name the nonterminals that are not preterminals "true-nonterminals" because they do not turn into terminals in one step. Mostly, one preterminal (except "Noun") will eventually turn into one word in the final sentence, while a true-nonterminal always turns into more than one words. So generally, true-nonterminals have more potential to generate long sentences.

Looking at all the rules that turn true-nonterminals into other symbols, we can find that only two rules turn a true-nonterminal to more than one true-nonterminals. they are

(i) `S` → `NP VP`,

(ii) `NP` → `NP PP`.

However, rule (i) is the only rule that turns `S` into other symbols—there is no other choice when `S` in the sentence. Things are different when it comes rule (ii). We actually can turn `NP` into `Det Noun` by another rule; and if we use rule (ii), `NP` becomes `NP PP`. Because `PP` can only turn into `Prep NP`, we have two `NP`s now: `NP Prep NP`. So it's like an explosion: one `NP` gives us two, and two `NP`s give us more. All thanks to the rule `NP` → `NP PP`.

(b)

(c)

(d)

(e)

3

(a)

4

(a)

5

(a)

6 Parse

(a)

7

(a) The parser does not always recover the original derivation that was "intended" by `randsent`. In fact, we can give an example of a sentence for which `parse` "misunderstood" and found an alternative derivation.

Our sentence was generated via the following tree:

```
(ROOT (S (N_clause that
                 (S (N_clause that
                             (S (NP (NP (NP (NP (Noun_pro Sally))
                                            (Conj and)
                                            (NP (Det a)
                                                (Noun (Adj (Adv really)
                                                           (Adj fine))
                                                      (Noun pickle))))
                                        (Conj and)
                                        (NP (NP (Det the)
                                                (Noun sandwich))
                                            (PP (Prep in)
                                                (NP (NP (NP (Noun_pro Xiaochen))
                                                            (PP (Prep in)
                                                                (NP (Noun_pro Sally))))
                                                        (PP (Prep with)
                                                            (NP (Det the)
                                                                (Noun sandwich)))))))
                                    (PP (Prep on)
                                        (NP (Det every)
                                            (Noun pickle))))
                                (VP (Verb wanted)
                                    (NP (Noun_pro Sally)))))
                    (VP (VP (Verb kissed)
                            (NP (Det a)
                                (Noun (Adj (Adv really)
                                           (Adj fine))
                                      (Noun sandwich))))
                        (Conj and)
                        (VP (Verb wanted)
                            (NP (Noun_pro Xiaochen))))))
         (VP (V_intran flew)))
      .)
```

(This problem is continued on the next page.)

However, `parse` found an alternate derivation via the following tree:

```
(ROOT (S (N_clause that
                   (S (N_clause that
                                (S (NP (NP (NP (NP (NP (NP (Noun_pro Sally))
                                                       (Conj and)
                                                       (NP (NP (Det a)
                                                               (Noun (Adj (Adv really)
                                                                          (Adj fine))
                                                                     (Noun pickle)))
                                                           (Conj and)
                                                           (NP (Det the)
                                                               (Noun sandwich))))
                                                   (PP (Prep in)
                                                       (NP (Noun_pro Xiaochen))))
                                               (PP (Prep in)
                                                   (NP (Noun_pro Sally))))
                                           (PP (Prep with)
                                               (NP (Det the)
                                                   (Noun sandwich))))
                                       (PP (Prep on)
                                           (NP (Det every)
                                               (Noun pickle))))
                                   (VP (Verb wanted)
                                       (NP (Noun_pro Sally)))))
                      (VP (VP (Verb kissed)
                              (NP (Det a)
                                  (Noun (Adj (Adv really)
                                             (Adj fine))
                                        (Noun sandwich))))
                          (Conj and)
                          (VP (Verb wanted)
                              (NP (Noun_pro Xiaochen))))))
         (VP (V_intran flew)))
      .)
```

(Note that the two trees are not equivalent.)

The reason for this is that the “correct” tree of a specific sentence could be ambiguous, depending on the grammar that was used. That is, if there is more than one unique tree that leads to the generated sentence, the parser could guess a tree that is different from what was actually used by the generator.

In our example, the sentence was composed of several subtrees first split using the `NP Conj NP` rule, and then later on split using the `NP PP` rule. However, the parser guessed that the sentence was composed of subtrees that consecutively split using the `NP PP` rule, and did not use the `NP Conj NP` rule to split until much later on.

(b)

(c)

(d) (i) Why is `p(best parse)` so small? Well, we can examine the probabilities used to generate this sentence, by first taking a look at the probabilities given from `grammar`.

$$\begin{aligned}\mathbf{Pr}[\text{this sentence}] &= \mathbf{Pr}[\texttt{ROOT}]\cdot\mathbf{Pr}[\texttt{S .}|\texttt{ROOT}]\cdot\mathbf{Pr}[\texttt{NP VP}|\texttt{S}]\cdot\mathbf{Pr}[\texttt{Det Noun}|\texttt{NP}]\cdot\\ &\quad \mathbf{Pr}[\texttt{the}|\texttt{Det}]\cdot\mathbf{Pr}[\texttt{president}|\texttt{Noun}]\cdot\mathbf{Pr}[\texttt{Verb NP}|\texttt{VP}]\cdot\\ &\quad \mathbf{Pr}[\texttt{ate}|\texttt{Verb}]\cdot\mathbf{Pr}[\texttt{Det Noun}|\texttt{NP}]\cdot\mathbf{Pr}[\texttt{the}|\texttt{Det}]\cdot\mathbf{Pr}[\texttt{sandwich}|\texttt{Noun}]\end{aligned}$$

Fair enough. Those are a lot of expressions multiplied together, which will probably produce a small number. But what does it equate, exactly?

$$\begin{aligned}\mathbf{Pr}[\texttt{ROOT}] &= 1\\ \mathbf{Pr}[\texttt{S .}|\texttt{ROOT}] &= 1/3\\ \mathbf{Pr}[\texttt{NP VP}|\texttt{S}] &= 1\\ \mathbf{Pr}[\texttt{Det Noun}|\texttt{NP}] &= 1/2\\ \mathbf{Pr}[\texttt{the}|\texttt{Det}] &= 1/3\\ \mathbf{Pr}[\texttt{president}|\texttt{Noun}] &= 1/6\\ \mathbf{Pr}[\texttt{Verb NP}|\texttt{VP}] &= 1\\ \mathbf{Pr}[\texttt{ate}|\texttt{Verb}] &= 1/5\\ \mathbf{Pr}[\texttt{Det Noun}|\texttt{NP}] &= 1/2\\ \mathbf{Pr}[\texttt{the}|\texttt{Det}] &= 1/3\\ \mathbf{Pr}[\texttt{sandwich}|\texttt{Noun}] &= 1/6\end{aligned}$$

$\implies \mathbf{Pr}[\text{this sentence}] = 5.144032922 \times 10^{-5}$

Let's call this number p, for simplicity purposes. Now let's think about why `p(sentence)` is equivalent to `p(best_parse)`. The probability of getting this specific sentence was p. Given that we have some $k \in \mathbb{N}$ number of possible parses of all of the possible sentences generated from `grammar`, the probability of getting the correct "best" parse of a sentence should *also* be p. That is, there should probably only be one possible parsing of each sentence, although some "types" of sentences could be more likely to appear than others.

Finally, we can think about why `p(best_parse|sentence)` = 1. If we run:

```
$ ./parse -c -g grammar
the president ate the sandwich .
```

we see that the number of possible parses of the sentence is 1. Therefore, there is only one possible "best" parse of the sentence, so the probability that the given parse is equivalent to the *only* best parse is 100%.

(ii)

(e)