University of Toronto, Department of Computer Science
CSC 485/2501F—Computational Linguistics, Fall 2018

Assignment 1

Due date: 14:10, Friday 5 October 2018, in tutorial.
Late assignments will not be accepted without a valid medical certificate or other documentation of an emergency.
This assignment is worth either 25% (CSC 2501) or 33% (CSC 485) of your final grade.

- Fill out both sides of the assignment cover sheet, and staple together all answer sheets (in order) with the cover sheet (sparse side up) on the front. (Don't turn in a copy of this handout.)
- Please type your answers/reports in no less than 12pt font; if you wish to include diagrams and tree structures in your reports, they may be drawn with software or neatly by hand.
- Any clarifications to the problems will be posted on the course website announcements page. You are responsible for taking into account in your solutions any information that is posted there, or discussed in class, so you should check the page regularly between now and the due date.
- What you turn in must be your own work. You may not work with anyone else on any of the problems in this assignment. If you need assistance, contact the instructor or the TA for the assignment.

1. Grammars [10 marks]

The grammar given below covers only **declarative** sentences, those that assert a proposition:

> People walk their dogs quickly in parks.

Two other kinds of sentences in English are *imperative* (commands):

> Walk your dogs quickly.

and *interrogative* (questions):

> Who walk their dogs quickly in parks?
> What will people walk quickly in parks?
> Where should people walk their dogs quickly?
> Should people walk their dogs quickly in parks?

Modify the grammar by adding or changing rules so that it accounts for these kinds of sentences. Your grammar should be reasonably principled and produce good sentences without producing bad ones. For example, it should **not** produce sentences such as these:

> *What people walk quickly in parks?
> *What should people walk their dogs quickly in parks?
> *Where walk their dogs quickly in parks?

(where the '*' indicates ill-formedness). Furthermore, you should only extend the lexicon of this grammar with the words, *who, what, where, should* and *will*.

Note: You do **not** need to account for verb endings due to person and tense (such as the difference between *walk*, *walked* and *walks*). We will see how to deal with these later. This grammar uses plural nouns so that you do not have to think about that. Nevertheless, some sentences will be nonsensical, although grammatical, e.g. *Parks walk dogs in people*. This is fine — what we are after here is grammaticality.

Note: This grammar is so simple that it cannot attach *in parks* to *their dogs* (i.e., people walking dogs that they keep in parks), although this is not how we would normally interpret this sentence in English anyway. This is also good enough for our purposes.

> Submit the source code of your grammar (in NLTK format) in a file called `q1.txt`.

Grammar:

S → NP VP
NP → N
NP → Det N
NP → Adj N
PP → P NP
VP → V Adv
VP → V NP Adv
VP → V NP Adv PP

Lexicon:

Det → *the* | *their* | *your*
Adj → *old* | *red* | *happy*
Adv→ *quickly* | *slowly*
N → *dogs* | *parks* | *statues* | *people*
V → *race* | *walk* | ~~*ate*~~ eat
P → *in* | *to* | *on* | *under* | *with*

2. Playing with NLTK [6 marks]

Try out the interactive recursive-descent and shift-reduce parser demos in NLTK (`nltk.app.srparser()`, `nltk.app.rdparser()`), and answer the following questions.

a. With the shift-reduce parser and its default grammar, parse the sentence *my dog saw a man with a statue in the park* as many different ways as you can. How many trees are there?

b. The default grammar with the shift-reduce parser includes the rule NP $\rightarrow$ NP PP, but that of the recursive-descent parser does not. Replace the NP rule of the latter with this rule:

   NP $\rightarrow$ NP PP | Det N

   and parse the default sentence. What happens? Now try:

   NP $\rightarrow$ Det N | NP PP

   Is this order change an adequate solution in general?

Submit your answers in a file called `q2.txt`.

3. A simple context-free grammar of English [30 marks]

Your task is to develop two context-free grammars for the chart parser in NLTK, each covering an interesting and not-entirely-trivial subset of English, along with a lexicon and a set of test sentences. In the first step, you'll build a base, which you'll then use for separate grammars in each of the next two steps. After testing your grammars, you'll write a report on what they can and can't do.

3.1 Very simple sentences

intransitive -> no object

To get started, begin with very simple sentences consisting of a subject noun phrase (NP), an intransitive verb in simple past tense (like *ate* or *arrived*), plus some modifiers. To begin introducing recursive rules in your grammar, allow the NPs to be modified by prepositional phrases as well as adjectives. You should be able to parse sentences such as the following (note that punctuation and sentence-initial capitalization should not be used in the test sentences; but words that are names can always be capitalized):

```
Nadia left immediately
the cat with the long soft fur slowly ate
she arrived
```

You should not, however, accept the following kinds of sentences. (Remember that '∗' means a sentence is ungrammatical.)

∗Nadia with the long soft fur slowly ate *(Can't attach a PP to a name)*
∗the cat with the tall her arrived *(Pronoun can't take an adjective)*

These are obviously not the only sentences your parser and grammar should accept or reject. Part of the point of the assignment is for you to decide what grammar you need in order for your parser to correctly accept or reject a range of constructions of this kind. Note that this does not mean simply listing lots of different words in the lexicon; rather, you need to think about similar *grammatical constructions*.

3.2 The auxiliary system

Agreement is one of the complex phenomena in natural language that makes writing good grammars difficult. This shows up in a particularly interesting way in the auxiliary system in English. In the next stage of development, extend your grammar to handle valid sequences of auxiliaries and verbs in English, such as those below.

```
Nadia will leave
Nadia has left
```

```
Nadia may have been leaving
*Nadia will left
*Nadia has could leave
*Nadia has had left
```

See the lecture slides and Jurafsky & Martin §12.3.6 for additional possible combinations of auxiliaries. Your analysis of the English auxiliary system must include the existential passive, which means that you will need to include the past participles of some transitive verbs in your lexicon as well.

Note: To keep things simple for the next part of the grammar in section 3.3, keep one rule `S -> NP VP` for sentences containing verbs in the simple past tense (without auxiliaries).

3.3 Subcategorization

Next, increase the coverage of your base grammar to deal with verbs other than those that are intransitive. A very important aspect of language in which one constituent places strong constraints on other constituents is the phenomenon of *subcategorization*. Subcategorization refers to the specification that a particular word places on the possible complements (objects) that it can occur with (see lecture notes #4).

Verbs take a wide variety of combinations of complements. Here are some example sentences your parser should be able to handle correctly; you should also think of other types of verbs (or get ideas from Jurafsky & Martin, §12.3.5) that you should be able to parse.

```
Nadia fondled the eggplant
the handsome poodle brought Ross to the autoclave
Nadia brought a cloth for the cheese
they told her to jump onto the elephant
she believed that Ross was already on the hovercraft
she really wanted help
she really aspired to help
cheese was always on the menu
the eggplant reminded Nadia of Ross
*Nadia found
*Ross brought to him
*they told to jump onto the elephant
```

There are many other complement possibilities (and impossibilities!).

Note: To keep your grammar a manageable size, do **not** try to combine the rules for the auxiliary system from section 3.2 with the rules for verbs with different subcategorizations. That is, **DO NOT** add grammar rules that generate auxiliary+verb constructions except for intransitives (which have the simplest of subcategorization requirements). Complements should only be generated for verb phrases in the active voice, simple past tense.

Note: You should use standard context-free rules with atomic categories, not rules with features, to account for subcategorization. We will look at features later on in the class.

3.4 Testing

Your grammar should handle all the test cases given in this handout, plus a few others, all of which are available at:
`http://www.cs.toronto.edu/~gpenn/csc485/A1-test.txt`,
and a list of the words that they use is available at:
`http://www.cs.toronto.edu/~gpenn/csc485/A1-vocab.txt`;
note that some words can have more than one syntactic category and hence will have more than one entry in the lexicon. Also, you will have to add words to this set for your own testing. In particular, several verbs appear in this file without all of their inflected forms. Your lexicon should add the missing forms, as well as all of the inflected forms for the verbs that you add.

In addition to the sentences given here, you will need to develop an additional set of your own test sentences to fully demonstrate that the grammar meets its specifications and to reveal over- and undergeneration — that is, types of sentences that it shouldn't accept but does, and types of sentences that it should accept but doesn't.

For greater clarity, it is *imperative* that you test your grammars on invalid sentences as well as valid sentences. Grammars, like software, need to be robust to both valid and invalid inputs. Explain *in detail* the steps you took during testing to ensure that both valid and invalid sentences have been accounted for. Note that we will compare what you say to the tests that you actually provide in your submission.

You will need to write testing code that uses NLTK to parse your test sentences with your grammar. A simple way to use NLTK's chart parser is demonstrated in the following:

```
import nltk

grammar = nltk.grammar.CFG.fromstring("""
S -> NP VP
PP -> P NP
NP -> DET N | N | NP PP
VP -> V NP |  VP PP
DET -> 'the'
N -> 'Nadia' | 'man' | 'eggplant'
V -> 'rewarded'
P -> 'with'
""")

sentence = nltk.tokenize.word_tokenize("""
Nadia rewarded the man with the eggplant
""")

parser = nltk.parse.BottomUpChartParser(grammar)

for t in parser.parse_all(sentence):
  print(t)
```

This produces two parses for the given sentence:

```
(S
  (NP (N Nadia))
  (VP
    (VP (V rewarded) (NP (DET the) (N man)))
      (PP (P with) (NP (DET the) (N eggplant)))))
(S
  (NP (N Nadia))
  (VP
    (V rewarded)
    (NP
      (NP (DET the) (N man))
      (PP (P with) (NP (DET the) (N eggplant))))))
```

Refer to the NLTK API documentation for details. Your test code will also need to read input and write output, as described in section 3.6.4 below.

Note: Your code must run on the `teach.cs.toronto.edu` servers. The command `python` runs an old version of Python (2.7.13) but with the current version (3.2.4) of NLTK. Your code for this assignment must use Python 3.5 (typed `python3.5`), which comes with NLTK 3.2.1.

3.5 Limitations

At this point, you'll be able to parse lots of interesting sentences. But there will be lots of sentences, even ones similar to those above, that you *won't* be able to accept or reject correctly, because your grammar will necessarily be limited. For example, you aren't asked to implement subject–verb agreement.

In your report, you should discuss *in detail* the kinds of limitations that still exist in your final grammar, including shortcomings revealed by your tests. This section of your report is not intended to cover every aspect of English that you can't deal with correctly (that would be a very long report). Rather, you should focus your report on constructions that are very similar to those you have been asked to deal with. For example, aspects of NPs and VPs would be reasonable to mention.

3.6 Implementation details

In order for the grader to be able to semi-automatically test your work, each file must have the exact name and format specified in the following subsections.

Some overall specifications for your files:

- The first line of each file must be a comment with your name, login ID, and student ID.

- Each grammar rule, lexical entry, or sentence must appear on a separate line in its appropriate file.
- The symbol `%` at the beginning of a line in your grammar, lexicon, and sentence files should indicate a comment.
- You should organize and comment your grammar and lexicon, just as you would your code, to make it easily understandable.

3.6.1 Grammar

Name of the file: `Grammar`
An example:

```
% Your name, login ID and student ID go here.
S -> NP VP
NP -> NPrp
NP -> Det N
% NLTK allows abbreviations like the following
VP -> V | V NP
VP -> V NP PP
PP -> P NP
```

The start symbol for the grammar must be S. Please use reasonable names for the other grammar symbols (nonterminals and parts-of-speech), following the conventions below. You might need symbols that aren't listed here—devise reasonable names for them based on your reading from the textbook or the terms we use in class.

Nouns: NP, N, NPrp (proper noun), NPro (pronoun), NDem (demonstrative pronoun)
Verbs: VP, V, Aux, Modal
Adjectives: AdjP, Adj
Prepositions: PP, P
Other: Det (article or determiner), Dem (demonstrative determiner), Adv (adverb)

It is particularly important that your three grammars be compatible with each other, in the sense that non-terminals refer to the same kind of word or phrase across all three grammars. This is because they will reside in this one file, and when we test your grammar, we will load all of this file at once.

3.6.2 Lexicon

Name of the file: `Lexicon`
An example:

```
% Your name, login ID and student ID go here.
Det -> 'a' | 'an'
N -> 'elephant' | 'rutabaga' | 'autopoiesis' | 'shot'
NPro -> 'i'
NPrp -> 'Nadia' | 'Marseilles' | 'Google'
V -> 'won' | 'smiled' | 'demanded' | 'shot'
```

Note: If a word has n possible parts-of-speech (lexical category designations), then it is listed n times in the lexicon, once for each part-of-speech, as with *shot* above.

3.6.3 Test sentences

Name of the file: `Positive`
An example:

```
% Your name, login ID and student ID go here.
Nadia won an elephant
I could have demanded a rutabaga
autopoiesis always reminded her of Marseilles
```

Note: The sentences should not contain any punctuation marks. Note that proper names are capitalized, but the first word of a sentence is not (unless it is a proper name).
In the same format, include files `Negative`, with negative examples that your grammar correctly excludes, `Overgen`, with ungrammatical sentences that your grammar incorrectly includes, and `Undergen`, with grammatical sentences that your grammar incorrectly excludes. Ungrammatical examples should not have asterisks — the file names make it clear whether they are ungrammatical.

3.6.4 Output parse trees

While it is not necessary to submit a file continaing your output parse trees, it should be possible for the TA to run your parser in order to regenerate them as necessary. The output of your parser should produce trees exactly in the pretty-print format that is provided by NLTK.

3.7 What to submit

3.7.1 On paper

Staple together (no paperclips or folded corners, please) the following items in the following order:

- The project cover sheet (attached to the back of this handout).
- A written report describing the limitations of your grammar (see section 3.5 above) and your testing strategy — in particular, cite specific instances of overgeneration and undergeneration. This report should be no more than one page.

3.7.2 Electronically

In addition to your paper submission, you must submit your grammar and your output electronically. Please include:

- All the input files (`Grammar`, `Lexicon`, `Positive`, `Negative`, `Overgen`, `Undergen`) that you used to test and evaluate your parser.

Note: You do *not* need to submit the actual code that you use to run your tests.

Submit all required files using the `submit` command on `teach.cs`:

```
% submit -c <course> -a A1 <filename-1>...<filename-n>
```

where `<course>` is either `csc485h` or `csc2501h`, and `<filename-1>` to `<filename-n>` are the n files you are submitting. Make sure every file you turn in contains a comment at the top that gives your name, your login ID on `teach.cs`, and your student ID number.

Grading scheme

We will test your grammar on the examples in this handout as well as on some held-out test sentences (i.e., sentences that you haven't seen). **We will test your grammars on your sentences using a computer. It is imperative that you follow the formatting requirements provided in this handout. Remarks will not be granted to work that did not adhere to these requirements.**

Grammar 1: simple sentences	2 marks
Grammar 2: auxiliaries and modals	2 marks
Grammar 3: subcategorization	2 marks
Testing (including output parses) by us and by you: meeting specifications; tests of overgeneration and undergeneration	20 marks
Your report: description of your grammars' limitations and your testing strategy	4 marks
Total	**30 marks**

CSC 2501 / 485, Fall 2018: Assignment 1

Family name: Shao Given name: Yuhan

Staple to assignment this side up

CSC 2501 / 485, Fall 2018: Assignment 1

Family name: Shao Given name: Yuhan

Student #: 1002281327

I declare that this assignment, both my paper and electronic submissions, is my own work, and is in accordance with the University of Toronto Code of Behaviour on Academic Matters and the Code of Student Conduct.

Signature: Yuhan Shao

Grade:

Q1	_____ / 10
Q2	_____ / 6
Q4	_____ / 30
TOTAL	_____ / 46