Mandatory assignment 3, INF2820, 2013

- **Deadline: April 4, 6:00 pm**
- **To be delivered in devilry.**
- **You should put your username in the comments of the submitted files, like `# Oblig 3, jtl`**
- **The mentioned files can be found at /projects/nlp/inf2820/cfg**

Exercise 1 – SR-parsing

a) (**10 points**) (Warm-up) At the exam you will not have access to a computer; you have to use pen and paper. One exercise could for example be:

> "Let G1 be the CFG:
>
> ```
> S -> NP VP
> VP -> V NP | V NP PP
> PP -> P NP
> V -> 'saw' | 'ate' | 'walked'
> NP -> PN | Det N | Det N PP | 'I'
> PN -> 'John' | 'Mary'
> N -> 'dog' | 'man' | 'telescope' | 'park'
> P -> 'in' | 'by' | 'on' | 'with'
> Det -> 'a' | 'an' | 'the' | 'my'
> ```
>
> Show how a shift-reduce parser for G1 will recognize the sentence
>
> 1) Mary saw a man in the park
>
> Use the format
>
> ```
> | Mary saw a man in the park
> Mary | saw a man in the park       SHIFT
> PN | saw a man in the park         REDUCE
> NP | Mary saw a man in the park    REDUCE
> ```
>
> You only need to show one successful recognition. You don't have to find more than one analysis, and you don't have to show unsuccessful choices and backtracking."

For your own benefit as an exam preparation, you are advised to try to solve this task by pen and paper. You get a solution to this exercise, however, by using the file `sr_recognize.py`, and running it with trace. You can thereby check the correctness of your solution. You could also "solve" the exercise by taking the trace output and remove some lines, and this is what you have to deliver. Trace level 1 says a little less than what we ask for and trace level 2 says a little more. The solution should be of the form

```
#  <>  Mary saw a man in the park
Mary  <>  saw a man in the park    SHIFT
PN  <>  saw a man in the park      REDUCE
NP  <>  saw a man in the park      REDUCE
```

Submission: As described.

b) (**20 points**) We also see that the recognizer backtracks. This becomes more visible if we trace a slightly ungrammatical string of words, like

2) Mary saw a man in park

Try this. The trace gets much longer since the recognizer performs an exhaustive search. We see the same effect if we try to parse the grammatical (1) with `sr_parser.py`.

In Oblig. 2 we saw how we could get a substantial speed-up for the recursive descent parser for grammars of the following form:

- The `rhs` contains no terminals (only zero or more non-terminals)
- The `rhs` contains exactly one terminal (and nothing more)

We will now see that we also can get a substantial speed up for the shift-reduce parser for the same class of grammars, with an additional restriction. Since SR-parsing is purely bottom-up, the first clause must read

- The `rhs` contains no terminals (only *one* or more non-terminals)

Given that the grammar has the specified form, we know that each shift has to be followed by a reduction, because a constellation like

```
NP som kjente | læreren som smilte sov
```

cannot result in success. There will be no later opportunity to reduce 'som'.

You shall modify the program `sr_recognizer.py` such that a shift is always followed by a reduce given that the grammar has the appropriate form. You may (but you don't have to) use the results from Exercise 4, Obligatory assignment 2, i.e. the `refine` procedure. Then modify `sr_parser.py` similarly. You do not have to worry about how the traces for the modified versions look (unless you accept the challenge to try to get them pretty!). Check on a couple of sentences that `sr_parser_refined.py` yields the same results as `sr_parser.py.`

Submission: The files `sr_recognizer_refined.py` and `sr_parser_refined.py.` Parsing examples that show that `sr_parser_refined.py` gives the same results as `sr_parser.py.`

c) (**10 points**) To get an understanding of how much we have gained, we will use the function `timing` which is included in `sr_parser.py`. (The file was updated 8 March to include this.) It can be called as for example:

```
>>> timing("""sr_parse(grammar,"Mary saw a man with a dog".split())""")
```

We shall compare the performance of the original `sr_parser.py` to `sr_parser_refined.py`. For both parsers see what happens

- with the example above and
- then when you add another PP, e.g. "Mary saw a man with a dog in the park"
- then when you add 2 PPs, 3 PPs and finally 4 PPs to the original sentence so there are 5 PPs altogether

Submission: The 5 example sentences used. Run times for the 5 examples and the two procedures in a 2*5 table which makes it clear where the different numbers come from.

d) (**10 points**) But even the refined parser may run slowly when the sentences get longer. First to repeat some notation try

```
>>> a="Mary saw a man"+3*" in the park"
>>> a
```

Then try

```
>>> timing("""sr_parse(grammar,("Mary saw a man"+3*" in the park").split())""")
```

and then replace successively 3 with 4, 5, …, 9

Submission: The run times in a table which makes it clear where the different numbers come from.

Exercise 2 – CFG (10 points)

You should extend your grammar from oblig.2, `pp.cfg`, to cover some more constructions. First extend it with coordination of S-s, VP-s and NP-s as in

a) Kari smilte og barnet sov.
b) Kari sov og smilte.
c) Ola og Kari smilte.

or

a) Kari smiled and the child slept.
b) Kari smiled and slept.
c) Ola og Kari smiled

Then include some simplified relatives as in

d) barnet som VP
e) barnet som NP TV

or

d) the child who VP
e) the child who NP TV

Call the extended grammar `my.cfg` and use `nltk.ChartParser()` to make a corresponding parser `my_parser`. How many analyses does the grammar ascribe to the sentence?

3) Ola som fortalte at Kari sov og smilte likte barnet og huset ved vannet
3) Ola who told that Kari slept and smiled loved the child and the house by the lake

Submission: The grammar `my.cfg`, answer to the question of how many analyses, the trees for the sentence.

Exercise 3 – CNF and CKY (20 points)

a) We want to use the CKY-algorithm. We must first turn our grammar into Chomsky normal form. Find a CNF grammar in CNF which covers the same constructions as `my.cfg`; call it `cnf.cfg`. The two grammars will not generate the same trees but they should generate equally many trees for a sentence and there should be a natural one-to-one correspondence between the trees. Test `cnf.cfg` and compare it to `my.cfg` on the example sentences (a-d) from oblig 2, exercise 2 and on sentence (3) from ex 2 above.

Submission: The grammar `cnf.cfg`. The trees generated for sentence (3) and an explanation of which tree corresponds to which tree generated by `my.cfg`.

b) We may now use CKY. First we should check that the grammar is of the right form. The NLTK grammar class has a function for this

```
>>> cnf_cfg.is_chomsky_normal_form()
```

Also check the original grammar `my.cfg`
Then use `cky.py` and `cnf.cfg` to construct CKY-tables.

Submission: CKY tables for sentence (d) from oblig-2:ex.2 and for sentence (3) above.
(At the exam you may be asked to construct such a table by hand.)

Exercise 4 Feature-Based Grammars (20 points)

We will start to experiment with feature-based grammars as described in section 9.1 in the NLTK book. The task depends on whether your my.cfg is for English or Norwegian:

English: You should extend your grammar with

- present forms of the verbs in addition to the past forms
- plural forms of the nouns and NPs in addition to the singular forms

The goal is to account for subject-verb agreement. Your grammar should include

Kari sells the house
the children eat apples
Kari ate an apple
the children ate an apple

but exclude

Kari sell the house
the children eats apples

and similarly for all verbs and NPs.

Norwegian: You should extend your grammar with masculine forms of nouns, determiners and adjectives in addition to the neutral forms. Your grammar should include

et barn smilte
en gutt smilte
et stort barn sov
en stor snill gutt ga et lite barn en bil

But exclude

en barn smilte
et gutt smilte
et stor barn sov
en stor snill gutt ga et liten barn en bil

and similarly for all determiners, nouns and andjectives. Include at least 10 common nouns that have a gender different from neuter.

Your grammar should be stored under the name `my_feature.fcfg`. To test it you may use

```
>>> from nltk import load_parser
>>> cp = load_parser('file:<path to your grammar>')
>>> for t in cp.nbest_parse("et barn smilte".split()): print t
```

Submission: `my_feature.fcfg` together with an example run for the positive and negative example sentences above.