# NLMI 2015: Project Exercises
# Part B

The goal of this project is to consider how grammar transformations affect the accuracy of a PCFG parser. This project has two steps, in the first one you will deal with basic binarization of a grammar, in the second one you will implement horizontal and vertical Markovization plus (optionally) consider a transformation of your choice.

## 1 Data and the Parser

You are provided with a basic PCFG parser which supports only binarized treebanks. It implements a fairly generic form of the CKY algorithm (incl. unary rules). The PCFG model is estimated on a given binarized treebank and then applied to a test file. PCFG model performs smoothing for generation of words given PoS tags, but no smoothing of inner rules. Assuming that you follow the conventions described below, de-binarization (and grammar annotation removal) is performed automatically by the parser. The parser reports evaluation results (bracketing precision, recall and F1 + exact match on sentences). You can also observe predictions of the parser on given sentences and compare them with annotation by linguists. The command line looks like:

```
java -ea -Xmx1G -cp path-to-bin nlp.assignments.parsing.SimpleParser
-train binarized-train-file -test not-binarized-test-file
```

where

- `path-to-bin` is the path to parser binaries (e.g., `bin/`),
- `binarized-train-file` is the binarized treebank file you will be producing in the assignment,
- `not-binarized-test-file` is one of the test sets (IMPORTANT: it should not be binarized).

The package includes example binarized files, see README for details.

## 2 STEP 1: Binarization

Perform binarization as described in class. Let's assume that the original treebank fragment (i.e. a sub-tree) looks like:

```
(NP (NNP Rolls-Royce) (NNP Motor) (NNPS Cars) (NNP Inc.))
```

After the transformation, it should be represented as

```
(NP (NNP Rolls-Royce) (@NP->_NNP (NNP Motor) (@NP->_NNP_NNP (NNPS
Cars) (@NP->_NNP_NNP_NNPS (NNP Inc.)))))
```

If you follow this convention, after parsing a sentence, the parser will remove nodes with labels beginning with "@" before evaluation, and attach their children so that the transformation is reversed. The resulting recovered tree will be evaluated against the test set. If you do not follow the convention, the reverse transformation will not succeed and the results will be nonsensical. You can see additional examples in `train-tiny-lossless.txt`.

Use `train20.text` for training the parser, and one of evaluation sets for evaluation. Specifically, the validation set should be used during the debugging stage, but final testing should be performed on the test set (`test20.txt`). The final results should be described in a very short accompanying report. The results you will obtain should be around 76.5% F1.

Provide your code which takes as an input non-binarized files and save the results of the binarization procedure in a separate file. Command line format:[1]

```
./b-step1 -input [non-binarized] -output [binarized]
```

[1] `b-step1` can be a bash/csh script file invoking a Java virtual machine or a python interpreter. If you are not familiar with scripts, it is sufficient if your java class accepts the specified parameters. Then you command line may look like: "`java -cp . BStep1 -input [non-binarized] -output [binarized]`." This applies to the second assignment as well.

## 3 STEP 2: Markovization

At this stage you will need to extend / transform the treebank grammar. Minimally you should implement horizontal and vertical Markovization discussed in class. Try at least 4 combinations of the horizontal order $h = \{1, 2\}$ with the vertical order $v = \{1, 2\}$ (the binarization implemented in stage 1 can be regarded as $h = \infty$ and $v = 1$). The results with $h = 2$ and $v = 2$ should be around 81% F1 and 29% exact match.

To encode Markovization use virtually the same notation as used in class, this is important as in this case, the parser will be able to perform the reverse transformation. Additionally to removing nodes with labels beginning with "@" (as described above), it also removes all material on node labels which follow their base symbol (cuts at the leftmost _, -, ^, or : character). Examples: a node with label `@NP->_DT_JJ` will be spliced out, and a node with label NP^S will be reduced to NP. See also `train-tiny-h2v2.txt` for some more examples of $h = 2$, $v = 2$ Markovization.

If you have more time and curious what kind of additional transformations might help, you are very welcome to implement some extra transformation(s), including transformations you may invent yourself.[2] You can consider, for example:

- Klein and Manning [1] for different types of transformations additionally to vertical and horizontal Markovization (partially discussed in class);
- Schlund et al. [2] for a new transformation which has not even been tried on English.

Or, you can come up with your own (even radically new!) ideas. IMPORTANT: (1) do not add any annotation to the ROOT symbol; (2) make sure that your annotation is included after '_' or '^' as described above; (3) the annotation should be done before binarization (unless you have good reasons to change the order).

As before, use the validation set for initial experiments and the final test for results you include in the report.

[2] We will not penalize you for not doing this. However, if other components of your assignment have some shortcoming, this can be used as an extra credit to compensate on this.

Command line format for Markovization:
`./b-step2 -h [number] -v [number] -input [non-binarized] -output [binarized]`

If you implemented an additional transformation method, please describe how to use it.

Another alternative bonus points you can get if you implement the parser itself. For pre-terminal rules you should rely on exactly the same smoothing methods you used for POS tagging ("task model"). The hardest part will be implementation of the CKY algorithm but, given that you have implemented Viterbi in Part A, it should be manageable. Moreover, you can rely on the pseudocode presented in the lecture. In this case, you can either implement scoring yourself or rely on an external script to compute R, P and F1 (e.g., *evalb*: http://nlp.cs.nyu.edu/evalb/).

The final paper, summarizing the two stages, submitted along with the last version of the code, should discuss your experiments (e.g., annotation strategies you tried) and findings (at most 4 A4 pages total). Please follow exactly the same procedure to submit your assignments (code and papers) and use the same format as was requested for the Part A of the class.

## References


[1] D. Klein and C. D. Manning, *Accurate Unlexicalized Parsing*. Proc. of Association for Computational Linguistics, 2003. http://acl.ldc.upenn.edu/P/P03/P03-1054.pdf

[2] M. Schlund, M. Luttenberger and J. Esparza. *Fast and Accurate Unlexicalized Parsing via Structural Annotations*. Proc. of European chapter of ACL, 2014. http://aclweb.org/anthology//E/E14/E14-4032.pdf