Practical Course in Natural Language Processing

Parsing of Probabilistic Context-Free Grammars

Boris Petrov*

June 15, 2026

1. Key Information

1.1. Integration into the Study Program

Scope: 0/0/4 SWS (6 ECTS, 180 hours workload)

Duration: 15 weeks

Modules and Examination:

Module	Course Name	Examination
INF-B-510	Practical Course in Natural Language Processing (6 ECTS, 180 hours, covers 50 % of the module, cannot be supplemented with another practical course)	oral (for practical course)
INF-MA-PR	Master's Practical Course in Natural Language Processing (6 ECTS, 150 hours, covers 50 % of the module)	colloquium (for practical course)
INF-VERT2	Advanced Artificial Intelligence (15 ECTS, 10 SWS, covers 40 % of the module, cannot be supplemented with another practical course)	oral (entire module)
INF-VERT6	Advanced Theoretical Computer Science (15 ECTS, 10 SWS, covers 40 % of the module, cannot be supplemented with another practical course)	oral (entire module)

Distinction:

- The practical course does not fit into the module INF-D-940, as the module is divided into two parts (2 SWS practical course/etc. + 2 SWS seminar). There is an option to complete the practical course with reduced scope (2 SWS).

*Based on previous work by Richard Mörbitz.

- The practical course does not fit into the module INF-PM-FOR, as the module is divided into three parts (2 SWS lecture + 2 SWS exercise/seminar + self-study).

1.2. Prerequisites

- Basic theoretical understanding and fluency in using context-free grammars, as taught in the course *Formale Systeme* (Module INF-B-270).
- Fluency in a common programming language (C, C++, Python, Haskell, Rust, Java).

2. Task Description

Probabilistic Context-Free Grammars. In natural language processing, the membership of a word in a language is often not formulated as a truth value but as a real number between 0 and 1. This provides a *degree of membership* of the word to the language. For example, frequent linguistic constructions can be distinguished from rare ones. In a probabilistic context-free grammar (PCFG), each rule is assigned a real number between 0 and 1 (the *probability* of the rule), such that the sum of the probabilities of all rules with the same left-hand side equals 1. The probability of a derivation is obtained by multiplying the rule probabilities, and the probability of a word is obtained by summing the probabilities of all derivations of that word.

Parsing. Parsing refers to the computation of a syntactic structure of a given sentence w in a natural language $L(G)$. This syntactic structure is usually represented as a tree, the so-called *parse tree*. If the natural language $L(G)$ is given by a (probabilistic) context-free grammar G, then each parse tree corresponds to a leftmost derivation of w in G.

Probabilistic Parsing. Grammars for natural languages are usually *ambiguous*, i.e., there are multiple parse trees for the same sentence. The probability of a parse tree is the product of the probabilities of all rules used in its derivation. Probabilistic parsing resolves ambiguity using the probabilities of the parse trees. A typical problem is *best parsing*, finding a parse tree with the highest probability.

Treebanks and Grammar Induction. A treebank is a multiset of parse trees of a natural language. Treebanks are manually or semi-automatically created by linguists using special annotation tools from available texts (e.g., journal articles or parliamentary transcripts). They can be used, for example, to derive a probabilistic context-free grammar.

2.1. Tasks

The completion of 3 tasks marked as *optional* is mandatory.

(1) Implementation of an algorithm for grammar induction

- Input: Treebank in *Penn Treebank* (PTB) format:

  ```
  (ROOT (S (ADV-MO Also) (PTKNEG-NG not) (NP-SB (ART-NK the)
    (NN-NK police))) ($. .))
  ```

  Each tree is on its own line.

- Output: induced PCFG in the format of the Berkeley parser. Grammars induced from corpora in PTB format are terminal-separated, i.e., on the right-hand side of each rule, there is either a sequence of $n > 0$ non-terminals or exactly one terminal. Grammars are saved as follows, where `grammar` is the name of the grammar:
  - Each rule ρ of the form $A \to A_1 \dots A_n$ with $P(\rho) = w$ is saved in its own line in the file `grammar.rules`:

    ```
    A -> A_1 ... A_n w
    ```

    Non-terminals must not contain spaces. Spaces separate non-terminals, the arrow, and the probability w.
  - Each lexical rule ρ of the form $A \to$ `v` with $P(\rho) = w$ is saved in its own line in the file `grammar.lexicon`:

    ```
    A v w
    ```

  - Each terminal `v` is saved in its own line in `grammar.words`:

    ```
    v
    ```

- The probabilities of the grammar rules are determined by relative frequency estimation (RFE).

(2) Implementation of a parsing algorithm

- Input: binarized PCFG G, i.e., for each rule $A \to A_1 \dots A_n$, $n \leq 2$, and a sequence of preprocessed sentences as follows:
  - Each sentence is on one line.
  - Terminals (i.e., words and punctuation marks) are separated by spaces.
  - Special characters (parentheses, spaces within terminals) are replaced by escape sequences.
- Output: Sequence of trees in PTB format; for each input sentence w, the best parse tree of w in G or `(NOPARSE` w`)` if none exists.

The underlying algorithm will implement either bottom-up parsing (CYK) *or* best-first parsing (Knuth's algorithm).

(3) Implementation of corpus transformations

(3 a) Debinarization (removal of all nodes introduced by binarization with the form `A|<A1,A2,A3,...>`).

(3 b) Trivial *unking* (each word in the corpus that occurs at most n times is replaced by the symbol `UNK`). The unking must also be reflected in the parser.

(3 c) *(optional)* Binarization and markovization (insertion of additional nodes to achieve the binarization property) [4].

(3 d) *(optional)* Smoothing of the induced grammar (improving generalization by, e.g., replacing rare words with symbols [2, 1]).

(4) Implementation of parser optimizations

(4 a) *(optional)* Pruning (restricting the spans examined) through:

- Beam search on the entire agenda (only best-first parsing) *or*
- Beam search per cell of the chart (only bottom-up parsing),

Implementation through a beam of constant size (maximum number of spans considered simultaneously) *or* threshold beam (discarding spans whose probability falls below a threshold), or a combination of both.

(4 b) *(optional)* A^*-search (using the Viterbi inside/outside weights as a heuristic for selecting the next span from the agenda) [3]. The Viterbi outside weights are saved in a file `grammar.outside` in the following format: For each non-terminal `A` with outside weight `w`, the file contains a line of the form:

```
A w
```

In the case of bottom-up parsing, this task can also be combined with pruning, where the weight of an item used for the pruning decision is calculated as the product of the Viterbi outside weight of the non-terminal and the probability of the best derivation for the item.

2.2. Criteria for Awarding Credits

- Task (1): Grammar induction must produce reusable output.
- Task (2): Parser must output a parse of w in G (if one exists).
- Task (3): At least one of the two mandatory tasks must be completed.
- Colloquium / oral examination: must be completed.

To achieve good or very good grades, all mandatory requirements must be fully met within the given timeframe, and three of the optional tasks must be sufficiently or fully completed.

3. Submission

The submission should be done through e-mail (an archive or preferably a link to a Git repository) aimed at the latest for 12 p.m. (noon). You are allowed to add minor features and bug-fixes to your Git repository until 12 a.m. (midnight), which is the hard cut-off point. If you have submitted your repository for an earlier task, please still send an e-mail informing me that you have done the new task.

4. Schedule

In the following, LW stands for lecture week. A lecture week is a week of the semester in which lectures can take place.

21. 04. Introductory session/Tutorial 1 (Organizational matters, overview of PCFG parsing, induction mechanism), task assignment for Task (1).

1.–3. LW Independent work on Task (1).

8. 05. Submission.

12. 05. Tutorial 2 (Evaluation of the previous task, presentation of both parsing algorithms, data structures), task assignment for Task (2).

4.–8. LW Independent work on Task (2).

12. 06. Submission.

16. 06. Tutorial 3 (Evaluation of the previous task, presentation of corpus transformations), task assignment for Task (3).

23. 06. Tutorial 4 (Presentation of parser optimizations), task assignment for Task (4).

9.–13. LW Independent work on Tasks (3) and (4).

17. 07. Submission.

21. 07. Tutorial 5 (Evaluation of optional tasks, competition).

5. Competition

The competition will take place at the end of the 13th LW – after submission – among all submissions (participation is optional). The results will be presented and discussed in the subsequent, final tutorial.

- 2 categories: Accuracy and Speed.

- The jury may disqualify parsers (e.g., if only dummy trees are output to win the speed category).
- Ranking through automated tests.

6. Technical Implementation

The code must run in the PC pool APB/E069 (Ubuntu 22.04). An equivalent environment is available on the server `ganymed.inf.tu-dresden.de` (ZIH login via SSH). *This server is a login machine and should only be used for testing functionality, i.e., parsing a short sentence at most!* All steps to create the program should be executed via the command `make`. We provide a command-line interface (see Appendix A). The evaluation will be conducted through automated tests (integration test). To verify that the program call and output format are correctly implemented, we provide tests.

7. Use of AI

The use of AI/LLMs is allowed as long as it remains reasonable and that you let me know in what aspects you used it during submission. In any case, you own what you submit. You should understand it and be able to explain it during the oral exam.

A. Command-Line Interface

The submission includes an executable file named `pcfg_tool`. The implementation of the subtasks is invoked via subcommands (similar to `git`). Subtasks can be implemented as standalone programs; in this case, `pcfg_tool` must forward the command-line arguments and standard input of the respective subcommands to the corresponding programs.

Some arguments only apply to optional tasks; these are indicated in parentheses. If an optional task has not been solved, the program should terminate with status 22 when arguments related to this task are present. The parser should run without optimizations by default and assume no corpus transformations other than binarization. Optimizations and corpus transformations are explicitly activated via flags.

Sequences of trees and sentences are always input and output line by line, i.e., there is exactly one object per line. The program output must preserve the order of the input objects.

```
pcfg_tool [COMMAND]
  Tools for PCFG-based parsing of natural language sentences.

pcfg_tool induce [GRAMMAR]
  Reads a sequence of constituent trees from standard input
  and outputs a PCFG induced from these trees to standard
  output. If the optional argument GRAMMAR is specified, the
  PCFG is instead saved in the files GRAMMAR.rules, GRAMMAR.lexicon,
```

```
  and GRAMMAR.words (see (1)).

pcfg_tool parse [OPTIONS] RULES LEXICON
  Reads a sequence of natural language sentences from standard
  input and outputs the corresponding best parse trees in PTB
  format or (NOPARSE <sentence>) to standard output. RULES and
  LEXICON are the filenames of the PCFG (see (1)).

  -p --paradigm=PARADIGM
    Parser paradigm (cyk or deductive).
    Default: Choice in (2).
  -i --initial-nonterminal=N
    Define N as the start non-terminal.
    Default: ROOT.
  -u --unking
    Replace unknown words with UNK (3 b).
  -s --smoothing
    Replace unknown words according to the smoothing
    implementation (3 d).
  -t --threshold-beam=THRESHOLD
    Perform beam search with threshold (4 a).
  -r --rank-beam=RANK
    Perform beam search with a beam of constant size (4 a).
  -a --astar=PATH
    Perform A*-search (4 b). Load the outside weights from the
    file PATH.

pcfg_tool binarise [OPTIONS]
  Reads a sequence of constituent trees from standard input
  and outputs the corresponding binarized constituent trees
  to standard output (3 c).

  -h --horizontal=H
    Horizontal markovization with H.
    Default: infinite (999).
  -v --vertical=V
    Vertical markovization with V.
    Default: 1.

pcfg_tool debinarise
  Reads a sequence of (binarized) constituent trees from
  standard input and outputs the original (non-binarized)
  constituent trees to standard output.
```

```
pcfg_tool unk [OPTIONS]
  Reads a sequence of constituent trees from standard input
  and outputs the trees obtained through trivial unking to
  standard output.

  -t --threshold=T
    Threshold of absolute frequency for unking.

pcfg_tool smooth [OPTIONS]
  Reads a sequence of constituent trees from standard input
  and outputs the trees obtained through smoothing to standard
  output (3 d).

  -t --threshold=T
    Threshold of absolute frequency for unking.

pcfg_tool outside [OPTIONS] RULES LEXICON [GRAMMAR]
  Computes Viterbi outside weights for each non-terminal of
  the grammar and outputs them to standard output. If the
  optional argument GRAMMAR is specified, the outside weights
  are saved in the file GRAMMAR.outside (4 b).

  -i --initial-nonterminal=N
    Define N as the start non-terminal.
    Default: ROOT.
```

References

[1] Peter F. Brown et al. "Class-based N-gram Models of Natural Language". In: *Comput. Linguist.* 18.4 (Dec. 1992), pp. 467–479. ISSN: 0891-2017. URL: http://dl.acm.org/citation.cfm?id=176313.176316.

[2] Daniel Dakota. "The Devil is in the Details: Parsing Unknown German Words". In: *Language Technologies for the Challenges of the Digital Age*. Ed. by Georg Rehm and Thierry Declerck. Cham: Springer International Publishing, 2018, pp. 23–39. ISBN: 978-3-319-73706-5. DOI: 10.5555/176313.176316.

[3] Dan Klein and Christopher D. Manning. "A* Parsing: Fast Exact Viterbi Parse Selection". In: *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*. 2003. URL: http://aclweb.org/anthology/N03-1016.

[4] Dan Klein and Christopher D. Manning. "Accurate Unlexicalized Parsing". In: *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*

- *Volume 1*. ACL '03. Sapporo, Japan: Association for Computational Linguistics, 2003, pp. 423–430. DOI: 10.3115/1075096.1075150.