Prof. Dr. Jasmin Blanchette
Dr. Martin Desharnais-Schäfer
Dr. Michael Kirsten
Elisabeth Lempa

Ludwig-Maximilians-Universität München
Institut für Informatik
Discussion on 12.11.2025
Homework due on 19.11.2025 at 16:00

Exercise Sheet 5 in

Scientific and Technical English for Computer Scientists

The exercise sheets consist of in-class exercises and homework. The in-class exercises take place in the second half of the lecture time slots. The homework, which is optional and ungraded, can be submitted via the "Homework" section in Moodle. The homework is subject to peer review.

Unless indicated otherwise, generative artificial intelligence assistants such as ChatGPT may be used, as long as you acknowledge how you use them as specified by the Institute's policy on plagiarism.[1] However, you may not use such tools to generate peer reviews for you. In addition, we strongly recommend that you do not use them to generate entire solutions, since this would defeat the purpose of the exercises.

In-class exercise 5-1 ***Roles of Words*** Read the following abstract[2] and identify the subjects, verbs, direct objects, indirect objects, complement, and adverbs.

> An LL(1) parser is a recursive descent algorithm that uses a single token of lookahead to build a grammatical derivation for an input sequence. We present an LL(1) parser generator that, when applied to grammar $\mathcal{G}$, produces an LL(1) parser for $\mathcal{G}$ if such a parser exists. We use the Coq Proof Assistant to verify that the generator and the parsers that it produces are sound and complete, and that they terminate on all inputs without using fuel parameters. As a case study, we extract the tool's source code and use it to generate a JSON parser. The generated parser runs in linear time; it is two to four times slower than an unverified parser for the same grammar.

Homework 5-2 ***Do Not Repeat Yourself*** The authors of the abstract presented in exercise 5-1 have a predilection for the word *grammar*. The following extract is from their paper's introduction. For each occurrence of the word *grammar* below, indicate whether it could advantageously be eliminated, either by replacing it with a synonym or by rewriting the sentence.

[1] `https://www.medien.ifi.lmu.de/lehre/Plagiate-IfI.pdf`

[2] Sam Lasser, Chris Casinghino, Kathleen Fisher, and Cody Roux, "A Verified LL(1) Parser Generator," *ITP 2019*, volume 141 of *LIPIcs*, pp. 24:1–24:18, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

> The tool has two main components. The first is a parse table generator that, when applied to a context-free grammar, produces an LL(1) parse table—an encoding of the grammar's lookahead properties—if such a table exists for the grammar. The second component is an LL(1) algorithm implementation that is parameterized by a parse table. By converting a grammar to a table and then partially applying the parser to the table, the user obtains a parser that is specialized to the original grammar.

Homework 5-3 ***From Passive to Active*** You might remember the following abstract[3] from Exercise Sheet 1. Identify the sentences written in the passive voice, and recast them into the active voice. For bonus points, identify the occurrences of *which* that could have been *that*.

> An algorithm is described which is capable of solving certain word problems: i.e. of deciding whether or not two words composed of variables and operators can be proved equal as a consequence of a given set of identities satisfied by the operators. Although the general word problem is well known to be unsolvable, this algorithm provides results in many interesting cases. For example in elementary group theory if we are given the binary operator $\cdot$, the unary operator $-$, and the nullary operator e, the algorithm is capable of deducing from the three identities $a \cdot (b \cdot c) = (a \cdot b) \cdot c$, $a \cdot a^{-} = e$, $a \cdot e = a$, the laws $a^{-} \cdot a = e$, $e \cdot a = a$, $a^{--} = a$, etc.; and furthermore it can show that $a \cdot b = b \cdot a^{-}$ is not a consequence of the given axioms.
>
> The method is based on a well-ordering of the set of all words, such that each identity can be construed as a "reduction", in the sense that the right-hand side of the identity represents a word smaller in the ordering than the left-hand side. A set of reduction identities is said to be "complete" when two words are equal as a consequence of the identities if and only if they reduce to the same word by a series of reductions. The method used in this algorithm is essentially to test whether a given set of identities is complete; if it is not complete the algorithm in many cases finds a new consequence of the identities which can be added to the list. The process is repeated until either a complete set is achieved or until an anomalous situation occurs which cannot at present be handled.
>
> Results of several computational experiments using the algorithm are given.

[3] Donald E. Knuth and Peter B. Bendix, "Simple Word Problems in Universal Algebras," *Computational Problems in Abstract Algebra*, pp. 263–297, Pergamon Press, 1970.