Aspects of Grammar Compression
Description for the General Public

Computer data grow larger every day—movies have higher resolution, mp3 higher bitrate, everybody has larger number of friends on Facebook, etc. All that has to be stored on hard drives (even if it is in a cloud—then it is just on someone else's drive), send and processed. For this reason the data is now routinely compressed. Various applications call for various compression methods and in this project we will investigate one of those—the *grammar compression*.

The name of the compression may seem a bit cryptic and it is hard to explain its roots in a couple of sentences, but it is very easy to describe how it works: the compressed file is given as a set of rules of the form '`letter` $\to$ `sequence of letters`' for instance: $M \to miSP, S \to NN, N \to ssi, P \to pi$, and a starting sequence, say M. To obtain the original file one needs to replace the letters according to the consecutive rules: first replace M by $miSP$ then all S by NN then each N by ssi and so on. In our example we obtain the word 'mississipi.' Our encoding does not look significantly smaller, but the compression works good for longer files and moreover we should have encoded the rules in a much more succinct way. The grammar compression is a base for, among others, LZW and LZ78 compression standards, which are used, for instance, in GIF and PDF file formats.

And how to construct such compressed representation? In an ideal case we would like to have the smallest one, but it turns out that it is *provably* infeasible to do so (within reasonable time bounds). On the other hand, there are several methods to compute reasonably small compressed files. In this project we will investigate the most popular of those and analyse how well and how bad they can compress files. We also want to propose new variants of the grammar compression which will avoid the pitfalls of the previous variants.

One can ask: since we cannot compute the best grammar compressed representation then why do we bother to use it in the first place? First, the known methods are not as bad and they produce reasonably small outputs. Second, the grammar compression has a tremendous advantage: we can process the compressed files relatively easily, without the need of prior decompression. Think about a movie: it is compressed, yet we can watch, rewind and fast forward it with ease. In the the grammar compressed files we can search, edit, cut and paste as if the file was not compressed at all, regardless of what type of file this is: movie, text files or presentations. Moreover, such methods turned out to be useful in science itself: there are numerous problems for which the best known solution is to compress them and then process the compressed representation. This type of computation occur for instance in algebra, more specifically—in equations in groups. Our project will deal with this type of problems as well—we will give new solutions to problems in groups based on this approach.

As explained above, we know fast algorithms for the compressed files. But maybe they can be even faster? Or maybe this is not possible, at all? In this project we will investigate such questions as well: the *lower bounds* are strict, mathematical arguments that something cannot be done better or faster. We will show lower bounds for various problems related to grammar compressed representations, especially the ones related to the computation in science.

Everything said above applies to grammar-compressed data treated as a string of letters. In some sense this is all that we need, as everything processed by computers is a string of symbols, in fact, zeroes and ones (bits). But encoding everything into sequences of bits can loose the important information and it is sometimes good to apply the compression on a higher level and only afterwards turn it into a sequence of bits. Formats of data other than sequences also have their own grammar compression methods, tuned to the specific setting, but here the situation gets complicated, as there are many design choices which influence the final outcome and it is not easy to judge, which is better and why. In the project we will investigate and compare the grammar compression for the simplest data type more complex than strings, the so-called trees. To this end we will compare how well they compress and what sort of operations we can perform on the compressed representations.