I think that everyone familiar with the following fact: with growing input size changes the algorithms that efficiently process the input. Mainly we are talking about string algorithms. There are several approaches that able to overcome the difficulty of growing input. The idea for the first one is very natural: efficiently process the input in file system instead of operational memory. The idea for the second one is less clear. Storing and processing the input as compressed representation. So we reduce input size and loss some information. Also in some areas compressed representation is more natural. For example in molecular biology.

Various compressed representations are known. For example, collage systems, straight-line programs, text compression using antidictionaries. In this paper we consider only straight-line programs, since it's very popular. Later I discuss reasons of the popularity.

Let us fix finite, nonempty alphabet Σ. A straight-line program, shortly SLP, is a sequence of assignments of the following form. Where S_1, S_2 etc. are rules and $expr_1, expr_2$ etc. are expressions. Either an expression is symbol from the alphabet or a concatenation of two rules. So an SLP is a context-free grammar in Chomsky normal form. We say that an SLP S derives a text S. It is convenient to associate an SLP with its parse tree. In this figure you can see an SLP that derives 7-th Fibonacci word. This figure show us why SLPs provide compression. For example, the rule F_4 occurs three times in the parse tree but there is exactly one expression that describes F_4 rule.

Let us emphasize main features of SLPs. There are three positive features: SLP is well-structured data representation, there is polynomial relation between the size of SLP that derives a given text and the size of LZ-dictionary for the same text (we discuss this feature in details later), there are classic string problems that have polynomial solution depending on size of SLPs. For example, pattern matching, longest common substring, etc. But there are two negative features: normally constants hidden in big-O notation for algorithms on SLPs are often very big and polynomial relation between the size of an SLP for a given text and the size of the LZ77-dictionary for the same text doesnt yet guarantee that SLPs provide good compression ratio in practice. Thus, a major questions is whether or not there exist SLP-based compression models suitable to practical usage. Namely in this paper we try to answer on the following questions: How difficult is it to compress data to an SLPrepresentation? How large compression ratio do SLPs provide as compared to classic algorithms used in practice?

Let us start from the first question. So we try to solve SLP construction problem. It looks too abstract. The following theorem sad that the problem of constructing a minimal size grammar generating a given text is NP-hard. Hence we should to look for approximate solution. One solution of the problem proposed by Rytter. This solution based on idea of factorization.

A factorization of a text S is a set of strings $F_1, F_2, \ldots, F_k$ such that $S = F_1, F_2, \ldots, F_k$. The LZ-factorization of a text S is a set of strings: $F_1, F_2, \ldots, F_k$ etc. This factorization naturally generates by LZ77-encoding algorithm. We present two examples of 7th Fibonacci word factorization. The first one is a simplest factorization (symbols decomposition), the second one is LZ-factorization.

Rytter reformulate SLP construction problem in the following way. Also he propose an algorithm that solves SLP construction problem. The algorithm have the following features: sequential factors procession and using AVL-trees as SLPs parse trees representation. The second feature guarantee estimation on height.

Example discussion. At the last step we have a problem. We need to concatenate the big rule F_8 and the small rule F_3. We can't perform this operation exactly since we break the tree balance. To save tree balance we should go down to a leaf at rightmost branch and then return. But during return we should apply AVL tree rotations that may change structure of the tree.

The following theorem is proved by W. Rytter and the algorithm is central part of the proof. Since we may use LZ-factorization as input to the algorithm then we can claim that there is polynomial relation between the size of SLP for a given text and the size of LZ-dictionary for the same text.

Let us consider AVL tree rotations. There are two type of rotations. It is important to emphasize that every rotation generate three new rules and destroy three rules at worst case. There is no problem if you use operational memory. But in case of huge input we store AVL tree in file system and get two problems: random access to rules (since we go down in tree), refreshing tree structure (since we generate many dead rules). So we must optimize access to rules.