Automata can be seen as models of computer systems (i.e., of programs). What is a model? Roughly speaking, this is a simplified image of a reality, which helps to present some issue. Models are used in many fields: we have models of airplanes, personality models, models of the universe, etc. Representatives of multiple fields of science work on models; computer scientists as well.

The goal of the project is to solve some problems in classical automata theory, which we may call "forgotten problems". Some time ago (e.g. 10, 20, or 50 years ago) these problems were considered as important, fundamental problems. However, these problems were difficult, so that no one could solve them. Simultaneously, it was possible to make advances in the theory in other directions, without solving these problems. For these reasons, people lost interest in these problems.

We are going to revive these problems and once again try to solve them. The hope is that a lot of interesting tools (theorems) was developed in the meantime. Thus, while the problems were too difficult for people trying to solve them, say, 20 year ago, they may be not so difficult now, when a lot of additional knowledge is available.

Notice that, on the one hand, both theory and practice develop well without solving the aforementioned problems. On the other hand, the problems concern basic properties of widely used objects, so it would be delightful to solve them.

Notice also that the project is a part of fundamental research, where we want to theoretically understand and explain some phenomena. It does not give any immediate practical, commercial applications.

Let us now be more specific on the problems we want to solve. There are four main research tasks in the project:

**Simplification of automata.** The first group of problems concerns questions of the form: provide an algorithm that given some more complicated automaton (e.g. a pushdown automaton) says whether there exists a simpler automaton doing the same (e.g. a finite automaton recognizing the same language), and finds it if it exists. Problems of this style can be considered for different kinds of automata. We want to provide such algorithms for most commonly used types of automata. In most cases either no algorithm is known, or we have only extremely slow algorithms. We remark, however, that designing algorithms is just a superficial goal; the real (hidden) goal is to deeply understand which automata can be converted to a simpler type.

**Equivalence problems.** In problems from this group the goal is to provide an algorithm that given two automata (of the same kind) decides if they are equivalent. To make this precise, we have specify what "equivalent" means, and there are multiple reasonable choices for that, but generally we rather mean here how the automata behave rather than how they look like. And this is the source of difficulty: the algorithm may receive two automata that look completely different, although they actually do the same. In the project we consider questions of this kind for pushdown automata (representing recursive programs) as well as for $\pi$-calculus terms (representing communicating concurrent processes). Again, depending on particular case, either no algorithm is known at all, or we only have extremely slow algorithms.

**Expressivity of higher-order pushdown automata.** Higher-order pushdown automata were introduced as a model of programs using higher-order recursion (i.e., recursion where functions can take other functions as parameters; these type of recursion appears in most modern programming languages). Although these automata turned out to be very useful, some basic problems concerning their expressivity are still unsolved. For example, there are two variants of these automata: standard and collapsible; the collapsible variant is assumed better, but we do not have any particular example which can be expressed by collapsible automata and (provably) cannot be expressed by standard automata. Proving that the automata really differ is one of goals of the project. Another question is: how these automata compare with context-sensitive grammars, one of the four basic types of grammars introduced by Chomsky in 1956?

**Automata over infinite words and trees.** Automata working over infinite words and trees can be used as models of programs that run forever (e.g. servers—these are not programs intended to finish a computation and then stop in a finite time). While words allow to encode a single computation, the branching nature of trees allows to encode all possible computations of a program. Such automata working for infinite time, especially those involving trees, are not yet well understood. We want to understand (and show decidability of) some fundamental problems concerning these automata.