

Research Statement

Yegor Bryukhov

The role of computing systems grows in our society with every year. It becomes increasingly important to ensure the safety and reliability of their operation. as the price of an error could be lives, expansive equipment or simply big money. It is especially true for critical applications like: financial systems, life support, power grid control, traffic control (either ground, underground, air or space), remotely controlled expensive equipment (space programs). The complexity of these systems also grows, probably exponentially. This creates a need for computer-aided methods of software engineering. It includes but not limited to proof assistants, logical frameworks and automatic provers.

At the same time, there are few mathematical results that used computers to check a multitude of simple facts. The number of those facts is intractable for a human being; even if these results were converted to a human readable form they could hardly be checked “by hand” reliably. And this is probably a tendency in mathematics, it is likely that there will be more and more results of such kind. Today many mathematicians do not consider such computer generated proofs acceptable but we are probably bound to see its wide acceptance sooner or later. Some day it will probably be considered a good taste to submit papers with proofs checked by one of the publicly recognized computer proof checkers (just like \TeX is a de facto standard today but it had a certain skeptical response at first).

“filling-in” or “filling in” ?

For a tool to be useful and usable in a pursuit of computer checked proofs, it should be capable of filling-in “simple” gaps in users’ reasonings and be truly interactive. Such tools (not necessarily interactive) are called proof assistants. To be truly interactive, a proof assistant has to have response time in fractions of a second most of the time. It has to fill reasoning gaps automatically so that user could concentrate on real issues rather than superficial ones. There are two kinds of gaps actually: trivial issues that won’t even be addressed in an informal proof, and not so simple issues that can be resolved by following some algorithm. There is no formal boundary between those kinds of gaps, of course. The former kind of gaps should be resolved by system automatically and instantaneously. The latter is less demanding, problems of that kind could take some time to be solved but it still has to take reasonable time, although everyone defines “reasonable” individually.

One particular type of problems that can and should be solved automatically is whether a certain arithmetical formula logically follows from given assumptions. Of

course, it is not decidable for Peano Arithmetic but it is decidable for Presburger Arithmetic (arithmetic of linear terms). Many mathematicians and computer scientists do not realize how important numbers are in their reasonings that they do every day. One uses numbers whenever (s)he needs to measure something, index something or use the principle of induction. So automation of integer reasoning is a very important feature of any proof assistant.

As a part of my PhD thesis I implemented integer number theory in the MetaPRL proof assistant and a few algorithms that automatically decide validity of arithmetical problems and generate their proofs. MetaPRL's mainstream logic is the Constructive Type Theory which has undecidable typechecking (unlike most of other proof assistants). As a result, all the algorithms I implemented generate hundreds of typechecking subproblems even for simple original problems. Those typechecking subproblems are trivial but to (automatically) close them computer needs to spend some time. Despite this obstacle my implementation is very competitive with implementations for proof assistants with decidable typechecking.

The second part of my thesis is devoted to automatic proof search for modal logics with justified common knowledge.

Why is it interesting?

Plato defines knowledge as *justified true belief*. The modal logic approach to knowledge, developed by Hintikka, captures the *true belief* components of Plato's tripartite definition. The *justification* component was introduced to formal epistemology by Artemov and Nogina. Traditional definition of "*A is common knowledge*" is "everybody knows that A and everybody knows that everybody knows, etc" which is formalized using fixed-point construct. Unfortunately this approach leaves no hope for automatic proof search, hence only model checking methods were developed until recent. Artemov and Nogina approach lifts the curse of fixed-point axiomatization and allows to find real proofs automatically. So I implemented such prover in the MetaPRL framework.

It seems that the practical complexity of formulas with justified common knowledge is higher than the complexity of formulas without common knowledge. In the future I want to address those performance issues. And compare several proof search techniques applied to logics with justified common knowledge. It would also be nice to complement them with a model checking tool.