CARMA Workshop on Computer-Aided Proof

University of Newcastle, Australia

5 – 7 June 2019

Contents

Introduction to Interactive Theorem Proving in Coq

Rajeev Gore (ANU)

I will give an introduction to interactive theorem proving using Coq by showing how to encode the natural numbers and the notions of even and odd numbers into Coq. I will show how to prove a simple theorem about the relationship between odd and even integers using Coq. I will then show how to specify a function for addition over the natural numbers and finally show how to turn that specification into an ML program to compute the sum of two natural numbers. I will not cover the mathematical foundations of Coq as this talk is aimed at people who have some mathematical background but who do not have any background in interactive theorem proving or program synthesis.

Modular Synthesis of Certifying STV Counting Programs

Rajeev Gore (ANU)

I will first explain STV counting and the parlous state of computer-counting code implemented by various Election Commissions from around Australia. I will then explain how we used Coq to specify a "vanilla" version of STV as a proof-calculus and used it to extract a computer program which not only counts votes according to this specification but also produces a certificate during the count. The specification of the certificate is derived from the counting rules. We have proved, in Coq, that if the certificate is correct with respect to its specification, then the result it encapsulates must be correct with respect to the relevant specification of STV. The certificate is designed so that an average third-year computer science student could write a computer program to check the correctness of the certificate. In particular, each political party could hire their own programmer to easily scrutiny the count produced by any computer program, including our own, that produces such certificates. The only caveat is that we require the publication of all ballots.

Formalized mathematics in the Lean proof assistant I

Robert Y. Lewis (Amsterdam)

Lean is a young proof assistant under development at Microsoft Research. It has attracted a growing community interested in formalizing results in pure mathematics. A number of groups, including the Formal Abstracts team in Pittsburgh and the Lean Forward team in Amsterdam, are exploring different ways to use Lean toward this goal.

The first talk in this series will introduce Lean from the bottom up. We will examine the system's language, logic, and metaprogramming capabilities, and see why it is a promising tool for these projects. We will touch on mathlib, the community-driven project to build a library of formalized mathematics in Lean.

Formalized mathematics in the Lean proof assistant II

Robert Y. Lewis (Amsterdam)

The second talk will present a Lean formalization of Ellenberg and Gijswijt's recent solution to the cap set problem. Their proof, resolving a combinatorial question that stood open for decades, was celebrated for its ingenuity and use of "elementary" methods. It is extremely rare for cutting-edge mathematics to be formalized in a proof assistant. Our formalization builds on mathlib, and we will discuss how various features of the library were used in our project. This talk will not assume familiarity with the cap set problem, but it will build on the first talk.

Proof synthesis and differential linear logic

Daniel Murfet (Melbourne)

Linear logic is a refinement of intuitionistic logic which, viewed as a functional programming language in the sense of the Curry-Howard correspondence, has an explicit mechanism for copying and discarding information. It turns out that, due to these mechanisms, linear logic is naturally related to constructions in linear algebra and calculus, whereas such connections are not as obvious for intuitionistic logic. I will explain these connections and why they point to a natural "derivative" of proofs (and programs) in linear logic, and how these derivatives lead to new approaches to the problem of synthesising proofs and programs.

Mechanising Computability and Kolmogorov Complexity in HOL

Michael Norrish (Data61, CSIRO)

I will describe ongoing work to render large chunks of the fundamental theories of computability and Kolmogorov complexity in HOL. Being a classical system, the HOL systems built-in notion of function does not correspond to the computable functions, so one has to carve out these specially. We show that the (partial) recursive functions correspond (in computational power) to the terms of the (untyped) lambda-calculus, and prove a number of standard results (e.g., Rices Theorem) about the computable functions. Attacking Kolmogorov complexity, we continue in similar vein, managing to dodge the standard assumption that the underlying computational model is the Turing Machine.

Calculation-Assisted Proof

Nicole Sutherland (Sydney)

We will discuss how mathematical software can be used to assist in proofs. We will consider the computer algebra software Magma as an example and investigate results in a number of papers were Magma has been used to aid a proof.

Theorem Proving and Program Development with Magma

Don Taylor (Sydney)

Using reflection groups and groups of Lie type as examples I will describe how the Magma Computational Algebra System can be used to prove theorems and develop algorithms.

Let P be a parabolic subgroup of a finite unitary reflection group G. Then the normaliser of P in G is the semidirect product of P and a complement H. For almost all G this can be proved without resort to computer-assisted calculations but for a small number of cases Magma was used to find a set of roots whose stabiliser is a complement.

Let G be a simply connected finite group of Lie type over a finite field of characteristic p and let $\rho : G \to GL(V)$ be a rational highest weight representation in charactersitic p. Given an element A in the image of ρ I will describe how Magma assisted the development of an algorithm which constructs a word g (in Chevalley normal form) in the Steinberg generators of G such that $\rho(g) = A$.

On average length of Game

Thotsaporn Thanatipanonda (Mahidol)

My family liked to play Monopoly back when I was a kid. But we never finished the game as it took too long. In this talk, we will analyze the average length of other simple probability games like Gamble ruin, Pile, Bingo, snakes and ladders, Camel up, etc. using Doron Zeilberger's favorite method, the method of "Guess and Check".

A Probabilistic Two-Pile Game

Thotsaporn Thanatipanonda (Mahidol)

We consider a game with two piles, in which two players take turn to add a or b chips (a, b are not necessarily positive) randomly and independently to their respective piles. The player who collects n chips first wins the game. We derive formulas for p_n, the probability of the second player winning the game by collecting n chips first for the cases a, b are both positive, $\{a, b\} = \{-1, 1\}$ and $\{-1, 2\}$. The symbolic sum plays a crucial role in this project.

Specifying and reasoning about operational semantics in the Abella theorem prover

Alwen Tiu (ANU)

Computation systems or semantics of programming languages and type systems can often be specified in a relational model, for example, as a state transtion system, or a deduction system. A widely used relational specification for modelling programming languages and type systems is the so-called operational semantics, e.g., Plotkin's structured operational semantics (also called the small-step semantics), and Kahn's natural semantics (the big-step semantics). A challenge in specifying operational semantics and reasoning about its properties in a proof assitant is how we represent variable binding constructs in programming languages, and more importantly, how we reason about these binding constructs. The Abella theorem prover is an interactive proof assistant based on intuitionistic logic, enriched with facilities to represent syntax with bindings (called lambda-tree syntax), and a nominal quantifier to model the dynamics of name bindings in proofs. In this talk I will give an overview of the foundational aspects of Abella, and give some examples of specifying operational semantics and reasoning about their properties.

References

1. The Abella theorem prover source code: http://abella-prover.org/
2. A tutorial paper on Abella: https://doi.org/10.6092/issn.1972-5787/4650

Experimental Mathematics: a Brave new Paradigm

Doron Zeilberger (Rutgers)

Jon Borwein, the great pioneer of Experimental Mathematics (and co-author, with David Bailey) of its bible, asserted that experimental mathematics is a paradigm shift, but even he still adhered to the old orthodoxy of proof-centric mathematics. I will describe some of the facets of experimental mathematics, and try and predict how mathematics will look like in fifty years.

A case-study in experimental-yet-rigorous mathematics: Analyzing the running time of Quicksort

Doron Zeilberger (Rutgers)

The shoemaker's children go barefoot. The analysis of algorithms, that attempts to find explicit expressions, or failing this, upper bounds, for the running time of COMPUTER algorithms, until recently, was a branch of traditional, paper-and-pencil mathematics. But one can get so much further if the computer is used to analyze the algorithms it runs, all by itself.

In this case study, I will illustrate this by describing a computer-aided (or rather computer-generated) analysis of the run-time of the famous Quicksort algorithm.

[Joint work with Shalosh B. Ekhad.]