Proceedings of the Second Workshop on Modules and Libraries for Proof Assistants (MLPA-09)

Florian Rabe, Carsten Schürmann
(organizers)

July 15, 2010, Edinburgh

affiliated with the IJCAR and ITP conferences at the Federated Logic Conference

Over the last twenty years, users of **proof assistants** and automated theorem provers have created large libraries of formal proofs and mathematical knowledge. **Module systems** help with the tedious tasks of organizing, sharing, and maintaining **libraries**. In the view of the ever increasing complexity of this network of information, module systems offer many of the answers to the practical problems that proof assistant system developers face today and can therefore be seen as an emerging research for the automated deduction community.

The **MLPA workshop** aims to attract and bring together researchers and practitioners with background and experience in the design, implementation, and application of module systems from different logic-based systems such as theorem provers, proof assistants, and programming languages.

The **second MLPA workshop** will be held during the 2010 Federated Logic Conference. It is affiliated both with the International Conference on Interactive Theorem Proving (ITP) and the International Joint Conference on Automated Reasoning (IJCAR). The **program committee** consists of

- Stefan Berghofer, Technische Universitt München, Germany
- Derek Dreyer, Max Planck Institute for Software Systems, Germany
- Georges Gonthier, Microsoft Research, United Kingdom
- Zhaohui Luo, University of London, United Kingdom
- Till Mossakowski, DFKI Lab Bremen, Germany
- Scott Owens, University of Cambridge, United Kingdom
- Florian Rabe, Jacobs University Bremen, Germany (chair)
- Carsten Schürmann, IT University of Copenhagen, Denmark (chair)

The **program** will consist exclusively of **invited talks**.

Modules and records in Agda

Ulf Norell

Chalmers University

Abstract

In Agda, record types and modules are orthogonal features; record types are simple sigma types without any advanced features such as subtyping or row polymorphism, and modules are non-first class entities whose purpose is to manage names. I will show how modules and records play together to allow us to work with hierarchical structures without having to extend the language with more powerful features.

Large-scale proof and libraries in Isabelle

Gerwin Klein

University of New South Wales

Abstract

In this presentation I will give an overview on the experience on proof-reuse and proof-libraries in Isabelle/HOL in two areas. The first is the L4.verified project, a large-scale proof of implementation correctness on the C level of the seL4 Operating Systems Microkernel. The proof consists of 200,000 lines of Isabelle script, includes multiple frameworks and libraries, had used external developments in core areas, and has contributed libraries and components back into the Isabelle distribution. The second part of the talk will be about the experience so far in submissions and reuse in the Archive of Formal Proofs (AFP), an archive of user-contributed Isabelle developments and libraries.

Mizar Mathematical Library - a large repository of formalized mathematics

Andrzej Trybulec

University of Bialystok

Abstract

Mizar is a computer system verifying mathematical proofs translated to or written in the Mizar language. The Mizar Mathematical Library (MML) is the main repository of formalized and computer-checked mathematics. Over two hundred authors contributed to the MML, which at the present includes almost ten thousand definitions and over fifty thousand theorems with complete proofs. The Hahn-Banach theorem, the Bing and Nagata-Smirnow metrization theorems, the Jordan curve theorem, and the Brouwer fixed point theorem are among the most advanced theorems contributed. The goal of the talk is to describe semantic mechnisms: structures, attributes and registrations used to develop abstract mathematics in Mizar.

Proofs in structured specifications

Don Sannella

University of Edinburgh

Abstract

Work on structured algebraic specifications provide syntax and semantics for a module language for proof assistants, together with a rich theory giving useful connections to relevant notions including module refinement and transformation. Modules can be built using different logical systems, once appropriate connections between the logics in use are provided. This theory takes a mainly model-oriented point of view, and proofs turn out to be less straightforward than one might wish. I will give an overview of this body of work, with stress on the issues that arise in connection with proof.

Towards a broad spectrum proof certificate

Dale Miller

INRIA

Abstract

Proof assistants and automated theorem provers generally produce evidence of a successful proof in an assortment of (often ad hoc) formats. The extent to which one prover can understand and check such evidence from another prover is the extent to which they can successfully inter-operate. I will outline some recent work on providing a single proof system that can be tuned so that it can host proof systems such as sequent calculus, natural deduction, tableau, and tabled deduction for classical and intuitionistic logics. A proof checker for the hosting proof system can then immediately be a proof checker for all of these other proof systems. The hope is that such a flexible format for proof certificates can be the basis of a lingua franca among proof systems. Central to this work is the use of linear logic and focused proof systems.

A Bottom-Up Approach to Safe Low-Level Programming

Adam Chlipala

Harvard University

Abstract

It's hard to please everyone with a systems programming language. If we follow the usual rules of language design, every programmer will miss some feature and complain that some feature should have been left out. Safe programming languages rely on increasingly complicated (and often arbitrary-seeming) type systems and program analyses, and yet still these

features don't support full verification. What if we stopped trying to design a single language and instead let programmers implement language features as libraries? With traditional compiler architectures, it would be very hard to ensure that the different features played well together... but what if our platform is a proof assistant, not a traditional compiler? In this talk, I will discuss some preliminary experiments of that kind: implementing an open "systems language" via Coq metaprogramming libraries that build verified machine code programs.