Theorem Proving in Higher-Order Logic

Tobias Nipkow
Technische Universität München, Germany

The focus of this lecture series will be HOL, Church's higher-order logic, which is the core of many interactive theorem provers. We will cover the following topics:

Foundations: HOL contains only simply typed lambda terms over equality and implication and a few inference rules for these constants. This basis can be extended with arbitrary new constants. First we show how to define the usual logic connectives and quantifiers and how to derive their familiar inference rules. Then HOL is turned into a programming language: we show how recursive data types and recursive functions can be defined within HOL.

Implementation issues: Sound implementations of HOL enforce that every proof is a combination of axioms, previously proved lemmas and inference rules. Even rewriting, the most important proof tool in interactive theorem provers, has to prove every step. We present a new scheme for obtaining a highly efficient rewriter for unconditional rewrite rules by compiling them into code.

Proof by reflection: This is a technique for programming decision procedures for fragments of the logic *within* the logic. Combining this with the compiled rewriter yields efficient decision procedures whose correctness must be proved before they can be used. We demonstrate this approach with a decision procedure for a fragment of arithmetic.

No previous exposure to theorem provers is required. Familiarity with functional programming and the basics of first-order logic is assumed.

References

1. R. Bird. *Introduction to Functional Programming using Haskell*, Prentice Hall, 1998
2. M. Huth, M- Ryan. *Logic in Computer Science*, Cambridge University Press, 2000
3. L.C. Paulson. *ML for the Working Programmer*, Cambridge University Press, 1996
4. S. Thompson. *Haskell: The Craft of Functional Programming*, Addison-Wesley, 1999