Computing, Solving, Proving: A Report on the Theorema Project (INVITED TALK)

Bruno Buchberger

Research Institute for Symbolic Computation
A4232 Schloß Hagenberg, Austria
<Bruno.Buchberger@risc.uni-linz.ac.at>

In an oversimplified and abstract view, given a logic (syntax, semantics, inference system) L and a knowledge base K of formulae in L,

- a "*computer*" for K enables the user to provide an expression (term, formula, program) T with a free variable x and a value v (from an appropriate domain) and "evaluates" $T_{x \leftarrow v}$ (T with v substituted for x) w.r.t. K in L,
- a "*solver*" for K enables the user to provide an expression T with a free variable x and produces (all) values v for which $T_{x \leftarrow v}$ holds w.r.t. K in L, and
- a "*prover*" for K enables the user to provide an expression T with a free variable x and generates a proof (or disproves) that, for all values v, $T_{x \leftarrow v}$ holds w.r.t. K in L.

Of course, computing, solving, and proving are mutually dependent. For example, a prover for a certain class of existentially quantified formulae, essentially, is a solver. Also, certain solvers may need proving potential as a subroutine and vice versa. Both, solvers and provers, can be described by expressions that are evaluated on (abstract) computers. Anyway, from the point of view of a user, computers, solvers, and provers are doing three different things with a given expression T.

Existing computer algebra systems are quite good in computing and, for certain K, are also amazingly good in solving (example: the "Solve" routines for systems of non-linear algebraic equations) but weak in proving. On the other hand, existing theorem proving systems, for certain K, are amazingly strong in proving but, usually, are weak in solving and computing. It is natural to strive for systems that are strong in computing, solving, *and* proving.

The *Theorema* project designed and directed by the author at the Research Institute for Symbolic Computation aims at providing a uniform (logic and software) frame L for computing, solving, and proving.

In order to close the gap between computing / solving (computer algebra systems) on the one side and proving (theorem proving systems) on the other side one could start from either of the two sides. In the *Theorema* project, we decided to start from an existing computer algebra system, namely Mathematica (version 3.0) and to add proving facilities.

In principle, one could of course take any of the existing computer algebra systems as a starting point for such a project. Here are the reasons why, after a profound analysis of the current computer algebra systems, we decided to base our work (at least in the prototype phase) on Mathematica 3.0:

- Essentially, the innermost part of *the Mathematica language is identical to higher-order equational logic*. As a consequence, Mathematica can be viewed as a "logic-internal" programming language and, thus, the language gap between computing and proving is closed in a natural way. In other words, formulae with an algorithmic content that are proved by one of our provers can be taken unchanged also as programs because the evaluation of such programs in Mathematica is nothing else than a restricted usage of the two basic inference rules of equational logic, namely substitution and replacement..
- The rule-based programming style of Mathematica lends itself to the implementation of provers and solvers.
- Mathematica has an attractive *arsenal of built-in functions* (like substitution, matching etc.) for formula manipulation, which are useful building blocks for programming the various provers in our system.
- Mathematica provides the *"notebook" facility* by which mathematical papers can be nicely structured and "active text" (programs that can be called from within the paper) can be intermingled with ordinary passive mathematical text.
- The new version 3.0 of Mathematica has highly sophisticated mathematical *typesetting facilities* even for executable formulae in the "input cells" of Mathematica "notebooks". In particular, Mathematica 3.0 also allows to change and extend its syntax and to attach user-defined semantics with syntactical notation.
- The text in notebooks can be structured in *nested cells* of arbitrary nesting depth. Such notebooks can be produced by Mathematica programs (e.g. our provers). This allows an elegant and novel way of presenting proofs in nested form.

Together, these features of Mathematica provide a good starting point for moving towards the final goal of the *Theorema* project, which is to provide a uniform (logic and software) system L in which a working mathematician, without leaving the system, can get computer-support while looping through all phases of the mathematical problem solving cycle: problem specification, definition of concepts, exploration of concepts, derivation of conjectures, proof of conjectures, extraction of algorithms from theorems, and presentation of results in interactive mathematical texts. In the talk we report on the present state and planned next steps of the *Theorema* project.