Maple as a Theorem Prover

November 23, 2006

Stephen Forrest

Department of Computing & Software
McMaster University

A theorem prover?

- How can a CAS like Maple possibly be regarded as a theorem prover?

- There are several ways:
  - Computation
    - Computation with assumptions
    - Computation as term-rewriting
  - The assume facility
  - Miscellaneous other features

Theorem Proving through Computation

- Many computations can be regarded as theorems.
- Any simplification or evaluation routine **f (x) ~> y** (e.g. `eval`, `normal`, `expand`, `simplify`, `radnormal`) can be regarded as a theorem asserting the equality of **x** and **y**.
- Domain of discourse is usually implicit in the choice of simplifier used (e.g. **evalc**). This makes it easy for the "wrong" command to be accidentally used.
- Some commands make use of assumptions.

Theorem Proving through Computation

- **Issues**
  - Domain of variables and operating theory implicitly specified
  - Implicit injections between theories: e.g. result from algebra used in an analytic computation
  - Soundness, robustness depend everywhere on correct implementation by programmers.

Theorem Proving through Computation

Issues (cont)

- Hard or impossible to see intermediate steps
- Conditions on results are inconsistently specified. Results can be provided:
    - With a side condition (proviso)
    - As a piecewise function
    - With no condition, provided “exceptions” occur on a set of measure zero (whatever this means!)
- Examples: `int(x^n, x)`:
    - what happens at n = -1?

Maple's Logic system

- In general, Maple's logic system is *ternary*: possible values are `true`, `false`, and `FAIL`.
- The value `FAIL` indicates that the computation of the boolean value was unsuccessful.
- In practice, large parts of Maple are two-valued (e.g. the type system).

Maple's Type System

- Maple types are predicates on expressions which are applied at runtime.
- The system has a hierarchy of *subtypes*,which means a value may have multiple types, e.g.

```
type( 1, integer );  # true
type( 1, positive ); # true
```

- Most types are “structural”, i.e. the typing rule is syntactic and doesn't depend on significant computation. (Not all, though!)

The assume facility

- Maple's assume facility allows checking of propositions subject to *assumptions*.
- Assumptions consist of boolean predicates or *properties.*
- Two main commands exist:
  - `is:` equivalent of ∀
  - `coulditbe:` equivalent of ∃
- General form (fV stands for “free variables”):
  - `is(p) assuming q` → ∀ fV(q,p) (q => p)
  - `coulditbe(p) assuming q` → ∃ fV(q,p) (q => p)

The assume facility

- All Maple types are automatically properties; however, we must now admit `FAIL` as a possible answer.
- Issues:
  - domain of variables still ill-defined
  - no way to “guide” computations other than providing assumptions
  - assumptions that are not understood are ignored
  - successes do not compose: getting true results from

    ```
    is(q) assuming p
    is(r) assuming q
    ```

    does not imply that `is(r) assuming p` will succeed.

References

- Maple 10 help system (see ?is, ?property)
- Weibel, Trudy, and Gonnet, Gaston. *An Algebra of Properties.* Proceedings of the ISSAC-91 Conference, Bonn July 1991, pp. 352-359.
- Weibel, Trudy and Gonnet, Gaston. *An assume facility for CAS with a sample implementation for Maple.* Conference Presentation at DISCO '92, Bath, England, April 14, 1992.
- Corless, Robert, and Monagan, Michael. *Simplification and the Assume Facility.* Maple Technical Newsletter, Vol. 1, No. 1, Birkhauser, 1994.