Computer Algebra Systems and Theorem Provers

Joe Hurd

Computer Laboratory
University of Cambridge

ARG Lunch
Wednesday 5 July 2006

Talk Plan

Introduction

- Computer algebra systems: Mathematica, Maple, etc.
- (Interactive) theorem provers: HOL, Isabelle, etc.
- Both process mathematical expressions, and can calculate either with numbers or symbolic terms.
- Both can be used to aid mathematicians:
  - Computer algebra systems are routinely used for testing conjectures at an early stage.
  - Theorem provers offer a gold standard of proof; especially important in cases where a purported proof is too long to be checked by humans (e.g., the four colour theorem, Kepler's conjecture).

Complementary Differences

- **Speed**
  - HOL is implemented in Standard ML, Mathematica in "an object oriented variant of C".
  - LCF style theorem provers impose a performance penalty: even term construction often requires object logic type checking.
  - Rewriting cannot compete with specialized algorithms.
  - Example: polynomial arithmetic (we'll see this later).
- **Reliability**
  - Most theorem provers emphasize logical soundness.
  - Most computer algebra systems will cut corners.
  - Example: when integrating x^n most computer algebra systems will return $x^{n+1}/(n+1)$, but this is wrong for $n = -1$.
  - Counterexample: Michael Beeson's MathXPert system for teaching students.

Another Difference

- **Usability**
  - Computer algebra systems are task-oriented, and are generally fully automatic.
  - Theorem provers support the task of interactive proof, but inexperienced users easily get stuck.
  - Is this a failure of theorem prover design, or does it just reflect the greater complexity of the task?

Computer Algebra Techniques in Theorem Proving

1. Use a computer algebra system as an oracle.
   - Needs careful handling to avoid unsoundness.
2. Use the computer algebra system to compute a witness for the problem, and then verify it in the theorem prover.
   - Sound, but not all problems fit into the model.
3. Implement computer algebra techniques as derived rules.
   - Sound, covers all problems, but might be inefficient.
4. Implement computer algebra algorithms and data structures as HOL functions, prove them correct and execute them in the theorem prover.
   - Sound and efficient (same complexity), but very hard.

Combination Projects

Using the categories of the previous slide:

1. OpenMath, MathML, MathWeb, and more
   Theorema (Buchberger et. al.) & Analytica (Clarke et. al.)
   Coding theory formalization (Ballarin & Paulson)
2. Primality certificates (Harrison & Théry, Caprotti)
3. Computer algebra techniques in HOL Light (Harrison)
   Computer algebra system in HOL Light (Kaliszyk & Wiedijk)
   Abstract algebra (the rest of this talk)
4. Buchberger's algorithm (Théry)
   Cylindrical Algebraic Decomposition (Mahboubi)

For many others look at the Calculemus conference proceedings.

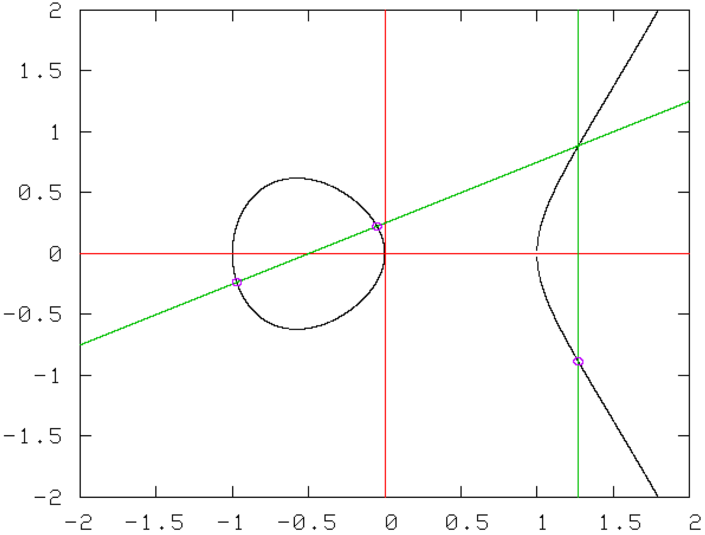
Elliptic Curves

- An elliptic curve over a field K is the set of points $(x, y) \in K^2$ satisfying a Weierstrass equation of the form

$$E: \; y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6$$

  where $a_i \in K$, plus a special point at infinity $\mathcal{O}$.
- It's possible to 'add' two points on an elliptic curve to get a third point on the curve.

The Elliptic Curve $y^2 = x^3 - x$: Addition



Verifying Elliptic Curve Addition

- Algebraic formulas are provided for adding and negating points.
- The goal is to show that elliptic curve addition forms an Abelian group.
  - Adding and negating points on E results in points on E.
  - Addition is associative: $(p_1 + p_2) + p_3 = p_1 + (p_2 + p_3)$.
  - Addition is commutative: $p_1 + p_2 = p_2 + p_1$.
- The rest of the talk will describe some computer algebra techniques implemented as derived rules that were developed to attack this goal.

Case Study: Point Doubling

Adding a point on the curve to itself results on a point on the curve:

Goal

$\forall$`e` $\in$ `Curve.` $\forall$`p` $\in$ `curve_points e.`
  `curve_double e p` $\in$ `curve_points e`

(13 symbols)

Case Study: Point Doubling

Stage 1: Expand definition of curve equation and point doubling

Goal

```
y' ** 2 + e.a1 * x' * y' + e.a3 * y' =
x' ** 3 + e.a2 * x' ** 2 + e.a4 * x' + e.a6
------------------------------------
0.  e ∈ Curve
1.  x ∈ e.field.carrier
2.  y ∈ e.field.carrier
3.  d ∈ field_nonzero e.field
4.  l = (3 * x ** 2 + 2 * e.a2 * x + e.a4 - e.a1 * y) / d
5.  m = (~(x ** 3) + e.a4 * x + 2 * e.a6 - e.a3 * y) / d
6.  x' = l ** 2 + e.a1 * l - e.a2 - 2 * x
7.  y' = ~(l + e.a1) * x' - m - e.a3
8.  d = 2 * y + e.a1 * x + e.a3
9.  y ** 2 + e.a1 * x * y + e.a3 * y =
    x ** 3 + e.a2 * x ** 2 + e.a4 * x + e.a6
```

(347 symbols)

Case Study: Point Doubling

Stage 2: Expand local definitions of all variables except the denominator d.

The goal is now of the form

$$\langle \textit{polynomial} \rangle [x, y] = 0 \implies \langle \textit{rational function} \rangle [x, y, d] = 0$$

(1,445 symbols)

Case Study: Point Doubling

Stage 3: Eliminate the division by d by lifting it to the top level and then expand the definition of d.

The goal is now of the form

$$\langle polynomial \rangle [x, y] = 0 \implies \langle polynomial \rangle [x, y] = 0$$

(2,690 symbols)

Optimization: When lifting $a/b + c/d$, must compute the polynomial gcd of b and d to keep the resulting term size down.

Elliptic Curve Gröbner Basis

- Want to replace the elliptic curve polynomial with a set of normalizing rewrites: a Gröbner basis.
- This is a trivial case of Buchberger's Algorithm.
- Give x a larger weight than y and write the equation as

$$x^3 = -a_6 + a_3 y + y^2 - a_4 x + a_1 xy - a_2 x^2 \qquad (*)$$

- Multiply everything out, replacing

$$x^n = x^3 x^{n-3} \qquad (n \geq 3)$$

and reducing x^3 with the simplifying rewrite $(*)$ above.

Elliptic Curve Gröbner Basis

- Optimization: Precompute

$$x^n = \langle \mathit{polynomial} \rangle \, [x, y]$$

  for all powers of n that are needed, simplifying the right hand side so that it has no powers of x larger than 2.
- For the point doubling running example, x^9 is needed.
- The right hand sides can get quite large: x^9 'simplifies' to a term with 5,000 symbols.

Case Study: Point Doubling

Stage 4: Replace the elliptic curve polynomial with the normalizing rewrites $x^i = \cdots$.

The goal is now of the form

$$\langle \mathit{rewrites} \rangle \implies \langle \mathit{polynomial} \rangle [x, y] = 0$$

(15,573 symbols)

Case Study: Point Doubling

Stage 5: Multiply out the polynomial, and reduce using the normalizing rewrites. Finally cancel terms to obtain the trival goal

$$0 = 0$$

That's the theory, anyway. Unfortunately, in practice the normalization takes way too long.

($>$300,000 symbols)

Tips for Efficient Handling of Large Terms in HOL

Or: Four things I wish I'd known when I started

1. Give the HOL pretty printer your own print functions (using `temp_add_user_printer`) to make the output as readable as possible. Cut off terms that are too big.
2. Make the simplifier work for you, by giving it simple rewrites and custom decision procedures for solving side conditions.
3. For complex normalization tasks add custom conversions to the simplifier, and stop it from descending into subterms matching P by giving it the null congruence rule $P = P$.
4. Writing correct normalization conversions is difficult: they tend to have corner cases that are hard to predict. The only way to be sure: repeatedly normalize until nothing changes!

Summary

- This talk has surveyed different ways of using computer algebra techniques and systems to support theorem proving, illustrating each combination method with past and current projects.
- A collection of proof tools to support abstract algebra in HOL was also presented, and demonstrated on a subgoal of the thorny problem to verify elliptic curve addition.
- Future work is clear: improve the proof tools until the whole verification can be completed. Suggestions welcome!