Formally Verifying Elliptic Curve Cryptography

Joe Hurd

Computer Laboratory
University of Cambridge

Åbo Akademi University
Thursday 5 October 2006

Talk Plan

Talk Plan

Verified ARM Implementations

- **Motivation:** How to ensure that low level cryptographic software is both correct and secure?

Verified ARM Implementations

- **Motivation:** How to ensure that low level cryptographic software is both correct and secure?
- **Project goal:** Create formally verified ARM implementations of elliptic curve cryptographic algorithms.

Verified ARM Implementations

- **Motivation:** How to ensure that low level cryptographic software is both correct and secure?
- **Project goal:** Create formally verified ARM implementations of elliptic curve cryptographic algorithms.
- The following elements are now in place:
  - A formal specification of elliptic curve operations derived from mathematics (Hurd, Cambridge). This talk!

Verified ARM Implementations

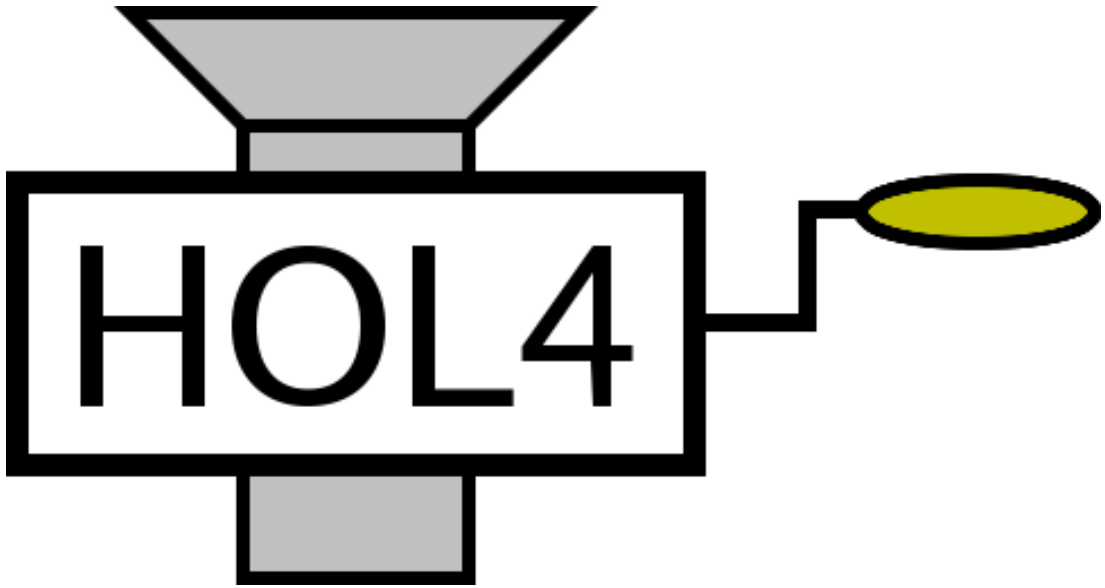
- **Motivation:** How to ensure that low level cryptographic software is both correct and secure?
- **Project goal:** Create formally verified ARM implementations of elliptic curve cryptographic algorithms.
- The following elements are now in place:
  - A formal specification of elliptic curve operations derived from mathematics (Hurd, Cambridge). This talk!
  - A compiler from higher order logic functions to a low level assembly language (Slind, Utah).

Verified ARM Implementations

- **Motivation:** How to ensure that low level cryptographic software is both correct and secure?
- **Project goal:** Create formally verified ARM implementations of elliptic curve cryptographic algorithms.
- The following elements are now in place:
  - A formal specification of elliptic curve operations derived from mathematics (Hurd, Cambridge). This talk!
  - A compiler from higher order logic functions to a low level assembly language (Slind, Utah).
  - A very high fidelity model of the ARM instruction set derived from a processor model (Fox, Cambridge).

Illustrating the Verification Flow

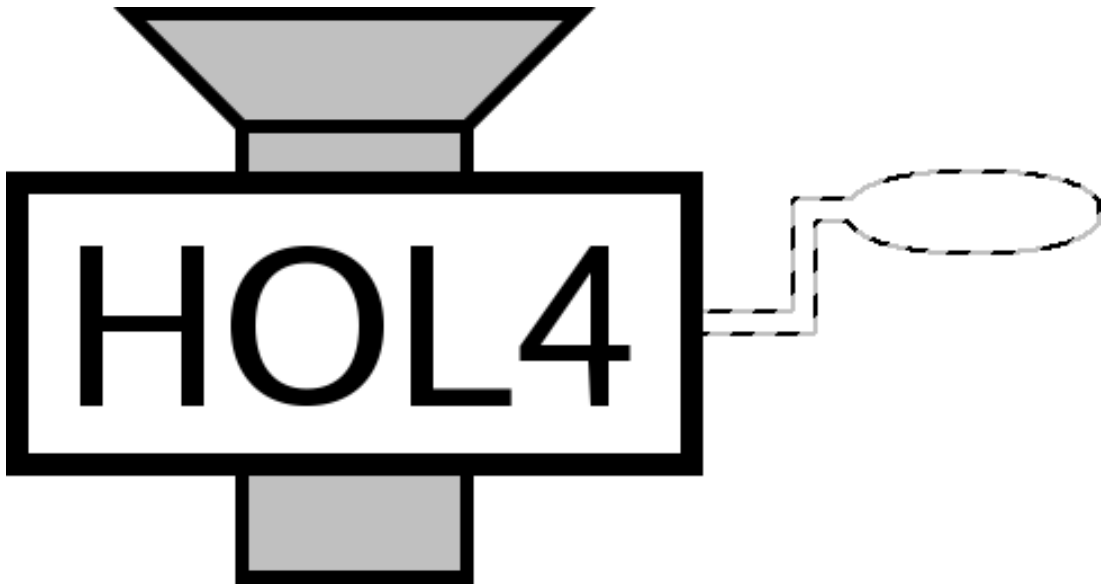
- Elliptic curve ElGamal encryption
- Key size = 320 bits



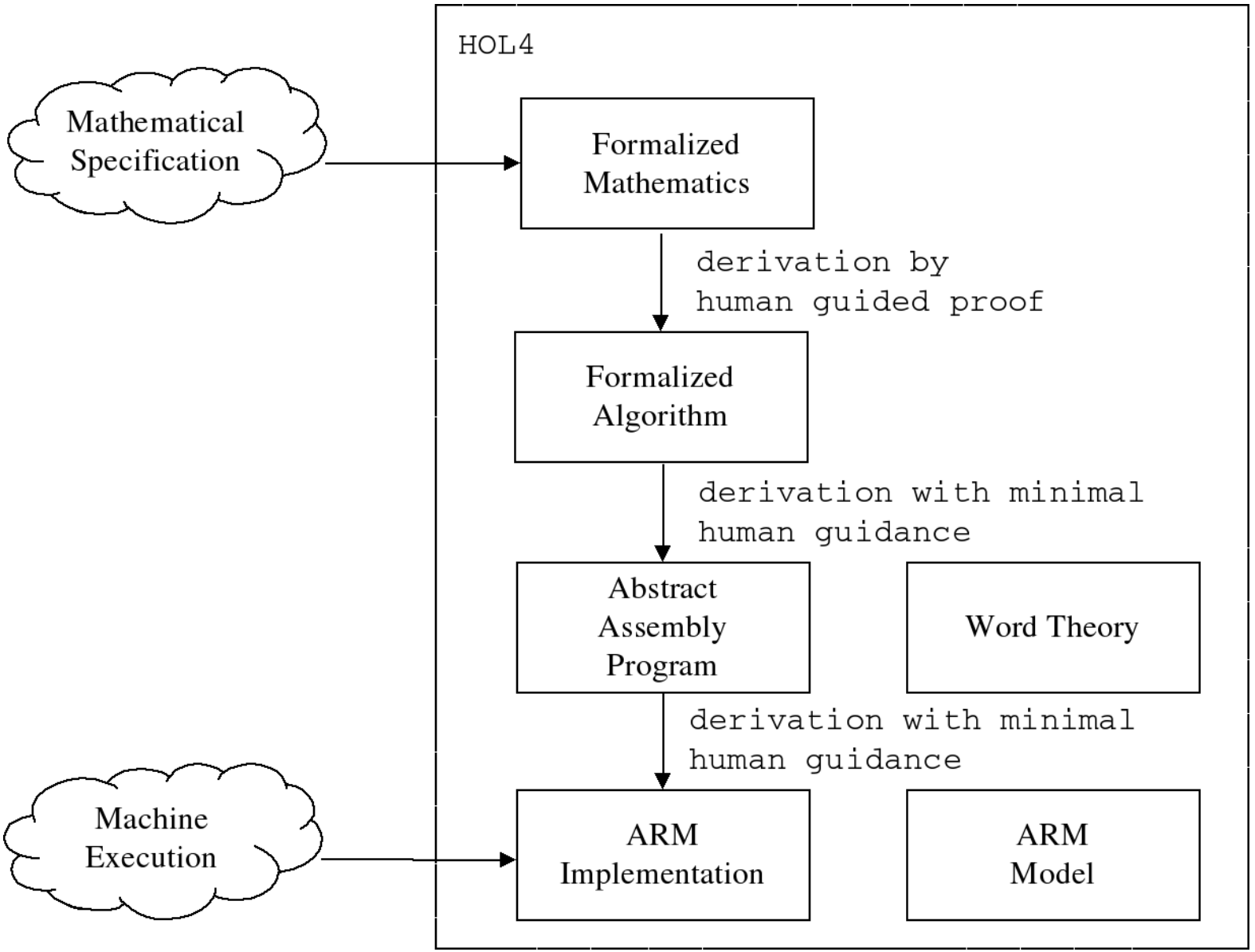
- Verified ARM machine code

Illustrating the Verification Flow

- Elliptic curve ElGamal encryption
- Key size = 320 bits



- Verified ARM machine code


HOL4
Mathematical Specification
Formalized Mathematics
derivation by human guided proof
Formalized Algorithm
derivation with minimal human guidance
Abstract Assembly Program
Word Theory
derivation with minimal human guidance
Machine Execution
ARM Implementation
ARM Model

Assumptions and Guarantees

- Assumptions that must be checked by humans:
  - **Specification:** The formalized theory of elliptic curve cryptography is faithful to standard mathematics. This talk!
  - **Model:** The formalized ARM machine code is faithful to the real world execution environment.

Assumptions and Guarantees

- Assumptions that must be checked by humans:
  - **Specification:** The formalized theory of elliptic curve cryptography is faithful to standard mathematics. This talk!
  - **Model:** The formalized ARM machine code is faithful to the real world execution environment.
- Guarantee provided by formal methods:
  - The resultant block of ARM machine code faithfully implements an elliptic curve cryptographic algorithm.
  - Functional correctness + a security guarantee.

Assumptions and Guarantees

- Assumptions that must be checked by humans:
  - **Specification:** The formalized theory of elliptic curve cryptography is faithful to standard mathematics. This talk!
  - **Model:** The formalized ARM machine code is faithful to the real world execution environment.
- Guarantee provided by formal methods:
  - The resultant block of ARM machine code faithfully implements an elliptic curve cryptographic algorithm.
  - Functional correctness + a security guarantee.
- Of course, there is also an implicit assumption that the HOL4 theorem prover is working correctly.

Assurance of the Specification

How can evidence be gathered to check whether the formal specification of elliptic curve cryptography is correct?

Assurance of the Specification

How can evidence be gathered to check whether the formal specification of elliptic curve cryptography is correct?

1. Comparing the formalized version to a standard mathematics textbook.

Assurance of the Specification

How can evidence be gathered to check whether the formal specification of elliptic curve cryptography is correct?

1. Comparing the formalized version to a standard mathematics textbook.
2. Deducing properties known to be true of elliptic curves.

Assurance of the Specification

How can evidence be gathered to check whether the formal specification of elliptic curve cryptography is correct?

1. Comparing the formalized version to a standard mathematics textbook.
2. Deducing properties known to be true of elliptic curves.
3. Deriving checkable calculations for example curves.

Assurance of the Specification

How can evidence be gathered to check whether the formal specification of elliptic curve cryptography is correct?

1. Comparing the formalized version to a standard mathematics textbook.
2. Deducing properties known to be true of elliptic curves.
3. Deriving checkable calculations for example curves.

This talk will illustrate all three methods.

Talk Plan

Elliptic Curve Cryptography

- First proposed in 1985 by Koblitz and Miller.
- Part of the 2005 NSA Suite B set of cryptographic algorithms.
- Certicom the most prominent vendor, but there are many implementations.

Elliptic Curve Cryptography

- First proposed in 1985 by Koblitz and Miller.
- Part of the 2005 NSA Suite B set of cryptographic algorithms.
- Certicom the most prominent vendor, but there are many implementations.
- Advantages over standard public key cryptography:
  - Known theoretical attacks much less effective,
  - so requires much shorter keys for the same security,
  - leading to reduced bandwidth and greater efficiency.

Elliptic Curve Cryptography

- First proposed in 1985 by Koblitz and Miller.
- Part of the 2005 NSA Suite B set of cryptographic algorithms.
- Certicom the most prominent vendor, but there are many implementations.
- Advantages over standard public key cryptography:
  - Known theoretical attacks much less effective,
  - so requires much shorter keys for the same security,
  - leading to reduced bandwidth and greater efficiency.
- However, there are also disadvantages:
  - Patent uncertainty surrounding many implementation techniques.
  - The algorithms are more complex, so it's harder to implement them correctly.

Elliptic Curve Cryptography: More Secure?

- This table shows equal security key sizes:

standard	elliptic curve
1024 bits	173 bits
4096 bits	313 bits

Elliptic Curve Cryptography: More Secure?

- This table shows equal security key sizes:

standard	elliptic curve
1024 bits	173 bits
4096 bits	313 bits

- But... there has been less theoretical effort made to attack elliptic curve cryptosystems.

Elliptic Curve Cryptography: A Comparison

Standard Public Key Cryptography

- Needed: a large prime p and a number g.
- Operation: multiplication mod p.
- Power operation: $k \mapsto g^k \bmod p$.

Elliptic Curve Cryptography: A Comparison

Standard Public Key Cryptography

- Needed: a large prime p and a number g.
- Operation: multiplication mod p.
- Power operation: $k \mapsto g^k \bmod p$.

Elliptic Curve Cryptography

- Needed: an elliptic curve E and a point p.
- Operation: adding points on E.
- Power operation: $k \mapsto p + \cdots + p$ (k times).

Cryptography Based On Groups

- The Discrete Logarithm Problem over a group G tests the difficulty of inverting the power operation:
  - Given $x, y \in G$, find a k such that $x^k = y$.

Cryptography Based On Groups

- The Discrete Logarithm Problem over a group G tests the difficulty of inverting the power operation:
  - Given $x, y \in G$, find a k such that $x^k = y$.
- The difficulty of this problem depends on the group G.

Cryptography Based On Groups

- The Discrete Logarithm Problem over a group G tests the difficulty of inverting the power operation:
  - Given $x, y \in G$, find a k such that $x^k = y$.
- The difficulty of this problem depends on the group G.
- For some groups, such as integer addition modulo n, the problem is easy.

Cryptography Based On Groups

- The Discrete Logarithm Problem over a group G tests the difficulty of inverting the power operation:
  - Given $x, y \in G$, find a k such that $x^k = y$.
- The difficulty of this problem depends on the group G.
- For some groups, such as integer addition modulo n, the problem is easy.
- For some groups, such as multiplication modulo a large prime p (a.k.a. standard public key cryptography), the problem is difficult.

Cryptography Based On Groups

- The Discrete Logarithm Problem over a group G tests the difficulty of inverting the power operation:
  - Given $x, y \in G$, find a k such that $x^k = y$.
- The difficulty of this problem depends on the group G.
- For some groups, such as integer addition modulo n, the problem is easy.
- For some groups, such as multiplication modulo a large prime p (a.k.a. standard public key cryptography), the problem is difficult.
- Warning: the number field sieve can solve this in sub-exponential time.

ElGamal Encryption (1)

The ElGamal encryption algorithm can use any instance $g^x = h$ of the Discrete Logarithm Problem.

1. Alice obtains a copy of Bob's public key (g, h).
2. Alice generates a randomly chosen natural number $k \in \{1, \ldots, \sharp G - 1\}$ and computes $a = g^k$ and $b = h^k m$.
3. Alice sends the encrypted message (a, b) to Bob.
4. Bob receives the encrypted message (a, b). To recover the message m he uses his private key x to compute

$$ba^{-x} = h^k m g^{-kx} = g^{xk-xk} m = m \ .$$

ElGamal Encryption (2)

Formalize the ElGamal encryption packet that Alice sends to Bob.

Constant Definition

```
elgamal G g h m k =
(group_exp G g k, G.mult (group_exp G h k) m)
```

This follows the algorithm precisely.

ElGamal Encryption (3)

Prove the theorem that Bob can decrypt the ElGamal encryption packet to reveal the message (assuming he knows his private key).

Theorem

```
⊢ ∀G ∈ Group. ∀g h m ∈ G.carrier. ∀k x.
    (h = group_exp G g x) ⟹
    (let (a,b) = elgamal G g h m k in
     G.mult (G.inv (group_exp G a x)) b = m)
```

This diverges slightly from the textbook algorithm by having Bob compute $a^{-x}b$ instead of ba^{-x}, but results in a stronger theorem since the group G does not have to be Abelian.

Talk Plan

Formalization in HOL4

- Formalized theory of elliptic curves mechanized in the HOL4 theorem prover.

Formalization in HOL4

- Formalized theory of elliptic curves mechanized in the HOL4 theorem prover.
- Currently about 4500 lines of ML, comprising:
  - 3500 lines of definitions and theorems; and
  - 1000 lines of custom proof tools.

Formalization in HOL4

- Formalized theory of elliptic curves mechanized in the HOL4 theorem prover.
- Currently about 4500 lines of ML, comprising:
  - 3500 lines of definitions and theorems; and
  - 1000 lines of custom proof tools.
- Complete up to the theorem that elliptic curve arithmetic forms an Abelian group.

Formalization in HOL4

- Formalized theory of elliptic curves mechanized in the HOL4 theorem prover.
- Currently about 4500 lines of ML, comprising:
  - 3500 lines of definitions and theorems; and
  - 1000 lines of custom proof tools.
- Complete up to the theorem that elliptic curve arithmetic forms an Abelian group.
- Formalizing this highly abstract theorem will add evidence that the specification is correct. . .

Formalization in HOL4

- Formalized theory of elliptic curves mechanized in the HOL4 theorem prover.
- Currently about 4500 lines of ML, comprising:
  - 3500 lines of definitions and theorems; and
  - 1000 lines of custom proof tools.
- Complete up to the theorem that elliptic curve arithmetic forms an Abelian group.
- Formalizing this highly abstract theorem will add evidence that the specification is correct. . .
- . . . but is anyway required for the formal verification of elliptic curve cryptographic operations.

Source Material

- The primary way to demonstrate that the specification of elliptic curve cryptography is correct is by comparing it to standard mathematics.

Source Material

- The primary way to demonstrate that the specification of elliptic curve cryptography is correct is by comparing it to standard mathematics.
- The definitions of elliptic curves, rational points and elliptic curve arithmetic that we present come from the source textbook for the formalization (*Elliptic Curves in Cryptography*, by Ian Blake, Gadiel Seroussi and Nigel Smart.)

Source Material

- The primary way to demonstrate that the specification of elliptic curve cryptography is correct is by comparing it to standard mathematics.
- The definitions of elliptic curves, rational points and elliptic curve arithmetic that we present come from the source textbook for the formalization (*Elliptic Curves in Cryptography*, by Ian Blake, Gadiel Seroussi and Nigel Smart.)
- A guiding design goal of the formalization is that it should be easy for an evaluator to see that the formalized definitions are a faithful translation of the textbook definitions.

Elliptic Curves

- An elliptic curve over the reals is the set of points (x,y) satisfying an equation of the form

$$E : y^2 = x^3 + ax + b \ .$$

- Despite the name, they don't look like ellipses!

Elliptic Curves

- An elliptic curve over the reals is the set of points (x,y) satisfying an equation of the form

$$E : y^2 = x^3 + ax + b \ .$$

- Despite the name, they don't look like ellipses!
- It's possible to 'add' two points on an elliptic curve to get a third point on the curve.

Elliptic Curves

- An elliptic curve over the reals is the set of points (x,y) satisfying an equation of the form

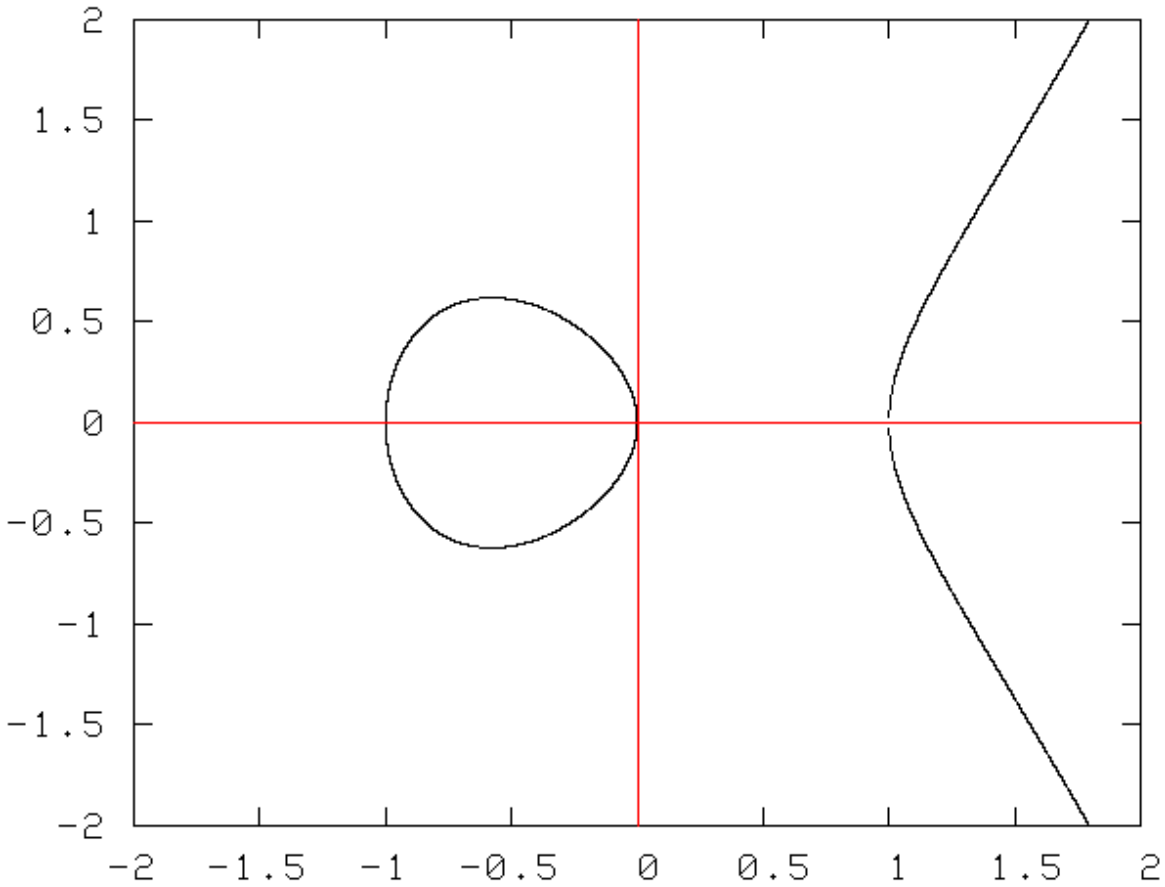
$$E : y^2 = x^3 + ax + b \ .$$

- Despite the name, they don't look like ellipses!
- It's possible to 'add' two points on an elliptic curve to get a third point on the curve.
- Elliptic curves are used in number theory; Wiles proved Fermat's Last Theorem by showing that the elliptic curve

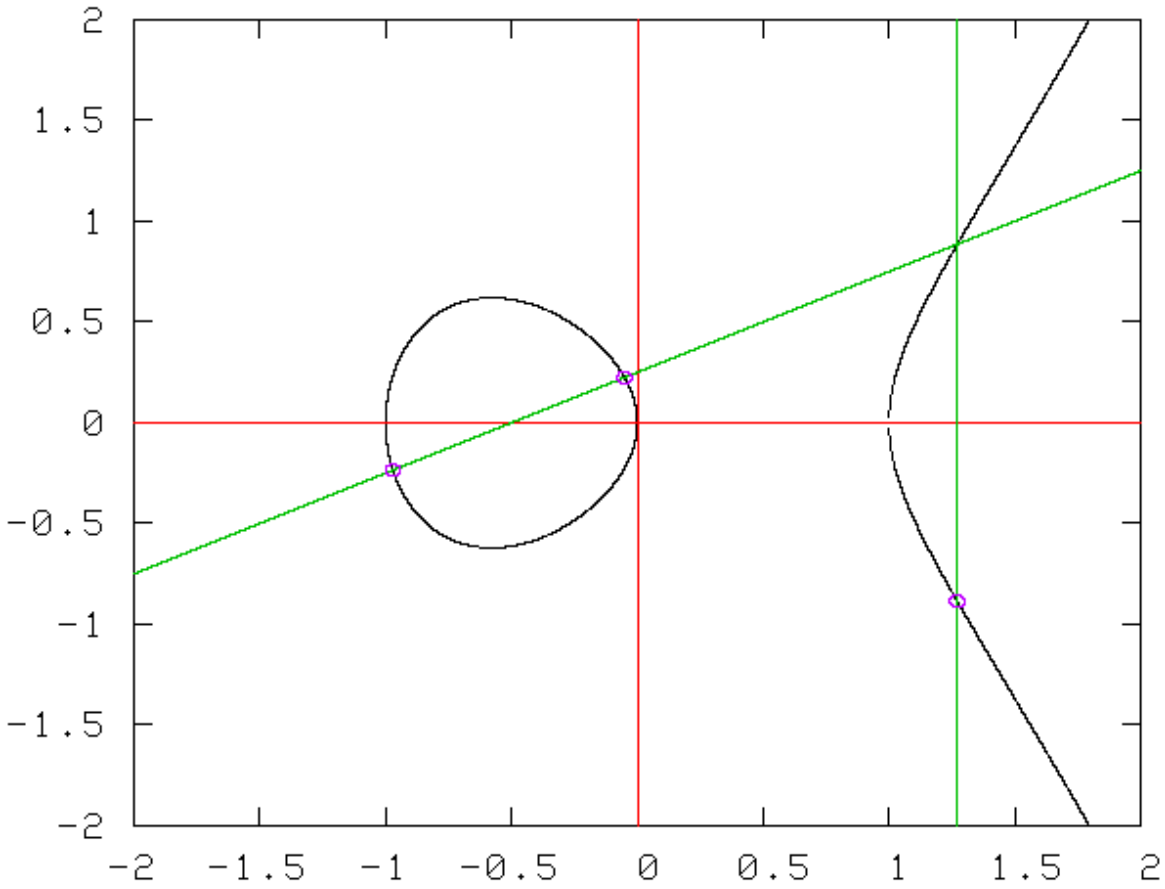
$$y^2 = x(x - a^n)(x + b^n)$$

generated by a counter-example $a^n + b^n = c^n$ cannot exist.

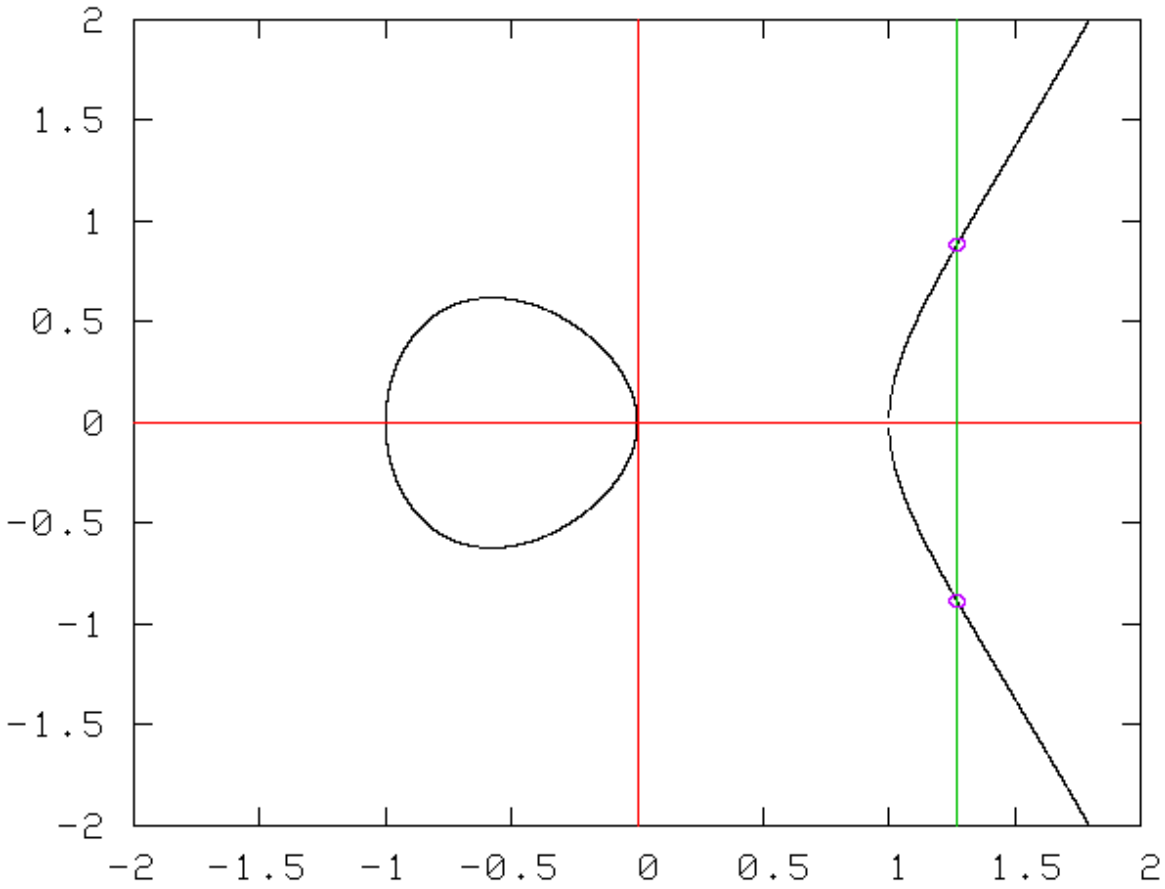
The Elliptic Curve $y^2 = x^3 - x$



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



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



Negation of Elliptic Curve Points (1)

Blake, Seroussi and Smart define negation of elliptic curve points using affine coordinates:

> *"Let E denote an elliptic curve given by*
>
> $$E : Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6$$
>
> *and let $P_1 = (x_1, y_1)$ [denote a point] on the curve. Then*
>
> $$-P_1 = (x_1, -y_1 - a_1x_1 - a_3)\ ."$$

Negation of Elliptic Curve Points (2)

Negation is formalized by cases on the input point, which smoothly handles the special case of $\mathcal{O}$:

Constant Definition

```
curve_neg e =
let f = e.field in
...
let a3 = e.a3 in
curve_case e (curve_zero e)
  (λx1 y1.
     let x = x1 in
     let y = ~y1 - a1 * x1 - a3 in
     affine f [x; y])
```

$$\text{“}-P_1 = (x_1, -y_1 - a_1 x_1 - a_3)\text{”}$$

Negation of Elliptic Curve Points (3)

The curve_case function makes it possible to define functions on elliptic curve points by separately treating the 'point at infinity' $\mathcal{O}$ and the other points (x, y):

Theorem

```
⊢ ∀e ∈ Curve. ∀z f.
    (curve_case e z f (curve_zero e) = z) ∧
    ∀x y. curve_case e z f (affine e.field [x; y]) = f x y
```

Negation of Elliptic Curve Points (4)

Negation maps points on the curve to points on the curve.

Theorem

$$\vdash \forall \texttt{e} \in \texttt{Curve}.\ \forall \texttt{p} \in \texttt{curve\_points e}.\ \texttt{curve\_neg e p} \in \texttt{curve\_points e}$$

Verified Elliptic Curve Calculations

- It is often desirable to derive calculations that provably follow from the definitions.
  - Can be used to sanity check the formalization,
  - or provide a 'golden' test vector.
- A custom proof tool performs these calculations.
  - The tool mainly consists of unfolding definitions in the correct order.
  - The numerous side conditions are proved with predicate subtype style reasoning.

Verified Calculations: Elliptic Curves Points

Use an example elliptic curve from a textbook exercise (Koblitz, 1987).

Example

```
ec = curve (GF 751) 0 0 1 750 0
```

Prove that the equation defines an elliptic curve and that two points given in the exercise lie on the curve.

Example

```
⊢ ec ∈ Curve
⊢ affine (GF 751) [361; 383] ∈ curve_points ec
⊢ affine (GF 751) [241; 605] ∈ curve_points ec
```

Verified Calculations: Elliptic Curve Arithmetic

Perform some elliptic curve arithmetic calculations and test that the results are points on the curve.

Example

```
⊢ curve_neg ec (affine (GF 751) [361; 383]) =
    affine (GF 751) [361; 367]

⊢ affine (GF 751) [361; 367] ∈ curve_points ec

⊢ curve_add ec (affine (GF 751) [361; 383])
               (affine (GF 751) [241; 605]) =
   affine (GF 751) [680; 469]

⊢ affine (GF 751) [680; 469] ∈ curve_points ec

⊢ curve_double ec (affine (GF 751) [361; 383]) =
    affine (GF 751) [710; 395]

⊢ affine (GF 751) [710; 395] ∈ curve_points ec
```

Doing this revealed a typo in the formalization of point doubling!

The Elliptic Curve Group

The (current) high water mark of the HOL4 formalization of elliptic curves is the ability to define the elliptic curve group.

Constant Definition

```
curve_group e =
<| carrier := curve_points e;
   id := curve_zero e;
   inv := curve_neg e;
   mult := curve_add e |>
```

Creating a mechanized proof that it actually is a group is ongoing, using computer algebra techniques to normalize the large polynomials that result.

Talk Plan

Computer Algebra Systems and Theorem Provers

- 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).

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.

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.

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.

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)

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)

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)

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.

Case Study: Point Doubling

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

Goal

```
∀e ∈ Curve. ∀p ∈ curve_points e.
  curve_double e p ∈ 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

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

(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

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

  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

$$0 = 0$$

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)

Simple Polynomial Normalization

- Even though 300,000 symbols is too much for the HOL theorem prover, it isn't a big problem for a computer algebra system.
- Would like a simple polynomial normalization algorithm.
  - Today will be used as an ML oracle.
  - One day could be formalized in HOL and proved correct.

Simple Polynomial Normalization

- Consider the following data structure for polynomials:

Type Definition

```
poly = Var of string
     | Sum of (poly,int) finite_map
     | Prod of (poly,int) finite_map
```

Simple Polynomial Normalization

- Consider the following data structure for polynomials:

Type Definition

```
poly = Var of string
     | Sum of (poly,int) finite_map
     | Prod of (poly,int) finite_map
```

- Example: $(2x + 3)^6$ is represented as

$$\text{Prod}\ \{\text{Sum}\ \{\text{Var}\ x \mapsto 2,\ \text{Prod}\ \{\} \mapsto 3\} \mapsto 6\}$$

- Note that numbers don't need a special constructor.

Simple Polynomial Normalization

- Simple normalization rules:

$$
\begin{aligned}
\text{Sum}\ (\{p \mapsto 0\} \cup M) &\longrightarrow \text{Sum}\ M \\
\text{Prod}\ (\{p \mapsto 0\} \cup M) &\longrightarrow \text{Prod}\ M \\
\text{Sum}\ (\{\text{Sum}\ M' \mapsto n\} \cup M) &\longrightarrow \text{Sum}\ ((n * M') \cup M) \\
\text{Prod}\ (\{\text{Prod}\ M' \mapsto n\} \cup M) &\longrightarrow \text{Prod}\ ((n * M') \cup M)
\end{aligned}
$$

Simple Polynomial Normalization

- Simple normalization rules:

$$\begin{aligned}
\text{Sum}\ (\{p \mapsto 0\} \cup M) &\longrightarrow \text{Sum}\ M \\
\text{Prod}\ (\{p \mapsto 0\} \cup M) &\longrightarrow \text{Prod}\ M \\
\text{Sum}\ (\{\text{Sum}\ M' \mapsto n\} \cup M) &\longrightarrow \text{Sum}\ ((n * M') \cup M) \\
\text{Prod}\ (\{\text{Prod}\ M' \mapsto n\} \cup M) &\longrightarrow \text{Prod}\ ((n * M') \cup M)
\end{aligned}$$

- One complicated normalization rule:

$$\text{Prod}\ (\{\text{Sum}\ S \mapsto n\} \cup P) \longrightarrow \text{Sum}\ (S^n * P)$$

where S^n is the multinomial

$$(x_1 + \cdots + x_m)^n = \sum_{k_1,\ldots,k_m} \binom{n}{k_1, \ldots, k_m} x_1^{k_1} \cdots x_m^{k_m}$$

Simple Polynomial Normalization

- These rules are sufficient to normalize polynomials.

Simple Polynomial Normalization

- These rules are sufficient to normalize polynomials.
- Though simple, they are efficient enough to prove the point doubling theorem in just a few seconds.

Simple Polynomial Normalization

- These rules are sufficient to normalize polynomials.
- Though simple, they are efficient enough to prove the point doubling theorem in just a few seconds.
- Notice that the bulk of the work is being done by the data structure, not the algorithm.

> *Show me your flowcharts and conceal your tables, and I shall continue to be mystified. Show me your tables, and I won't usually need your flowcharts; they'll be obvious. [Brooks, 1975]*

Talk Plan

Summary

- This talk has described a theory of elliptic curve cryptography mechanized in the HOL4 theorem prover.
- Assurance is needed: the formalized theory will be used to write specifications for verifying ARM implementations of elliptic curve cryptography.
- There's still work to be done combining computer algebra techniques with high assurance theorem provers.