Towards abstract and executable multivariate polynomials in Isabelle

Florian Haftmann

Institute for Informatics
TU Munich

Andreas Lochbihler

Institute of Information Security
ETH Zurich

Wolfgang Schreiner

RISC
Johannes Kepler University Linz

Isabelle 2014

Motivation

A cultural „gap“ between two communities.

- Theorem proving:
  - Sound formal development of theories on top of a small trusted kernel.
  - Computations reduced to logical inferences.
  - Correct but inconvenient to use and painfully slow.
- Computer algebra:
  - Elaboration of mathematics by paper-and-pencil or TP software.
  - Separate implementation in mathematical software systems.
  - Convenient to use and reasonably fast but highly untrustworthy.

How can we bridge this gap?

Our Starting Point: Polynomial Algebra

An Isabelle package in which the working mathematician can develop

- Mathematical theories based on an abstract view of polynomials.
  - Type-checked definitions and theorems.
  - Computer-supported/mechanically verified proofs.
- Algorithms based on the defined mathematical notions.
  - Executable with „reasonable“ efficiency (rapid prototyping).
  - Formal specification and computer-supported verification.

A single computer-supported formal framework for proving and computing with (multivariate) polynomials.

Polynomials

What is the polynomial written as $2x^3 - 5x + 7$?

- Traditional: the symbolic expression itself.

$$\left(\sum_{i=0}^{n} a_i x^i\right) \cdot \left(\sum_{j=0}^{m} b_j x^j\right) = \sum_{k=0}^{m+n} \left(\sum_{i \in \mathbb{N}_0, j \in \mathbb{N}_0}^{i+j=k} a_i \cdot b_j\right) \cdot x^k$$

- Computer science: an array $[7, 5, 0, 3]$

```
int[] mult(int[] a, int[] b)
{
  int m = a.length-1; int n = b.length-1;
  int[] c = new int[m+n+1];
  for (int i = 0; i <= m; i++)
    for (int j = 0; j <= n; j++)
      c[i+j] += a[i]*b[j];
  return c;
}
```

Two representations of a more fundamental concept.

Polynomials

The more fundamental concept is the modern view of polynomials.

- Polynomial: a function $[0 \mapsto 7, \dots, 3 \mapsto 3, 4 \mapsto \underline{0}, 5 \mapsto \underline{0}, \dots]$

  Let R be a ring. A (univariate) polynomial over R is a mapping $p : \mathbb{N}_0 \to R, n \mapsto p_n$, such that $p_n = 0$ nearly everywhere, i.e., for all but finitely many values of n.

- Elegant mathematics:

  $$\_ \cdot \_ : (\mathbb{N}_0 \to R) \times (\mathbb{N}_0 \to R) \to (\mathbb{N}_0 \to R)$$
  $$a \cdot b := k \in \mathbb{N}_0 \mapsto \sum_{i \in \mathbb{N}_0, j \in \mathbb{N}_0}^{i+j=k} a_i \cdot b_j$$

- Polynomial ring: $R[x]$

  The set of polynomials with $(+)$ and $(\cdot)$; variable x just denotes the polynomial $[0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 0, 3 \mapsto 0, \dots]$.

See e.g. [Winkler, 1996].

Multivariate Polynomials

What is a polynomial $3x^2y + 5yz$ in variables x, y, z?

- Polynomial: a function $[(1,1,0) \mapsto 3, (0,1,1) \mapsto 5, (0,0,0) \mapsto 0, \ldots]$

  An n-variate polynomial over the ring R is a mapping $p : \mathbb{N}_0^n \to R, (i_1, \ldots, i_n) \mapsto p_{i_1,\ldots,i_n}$, such that $p_{i_1,\ldots,i_n} = 0$ nearly everywhere.

- Polynomial ring: $R[x_1, \ldots, x_n]$

  The set of all n-variate polynomials over R; variable x_i denotes the polynomial $[\ldots, i \mapsto 1, \ldots]$.

- Isomorphism: $R[x_1, \ldots, x_n] \simeq (R[x_1, \ldots, x_{n-1}])[x_n]$.

  Recursive algorithms may be devised for many (not all) computational problems on multivariate polynomials.

  - Polynomial division is defined on $K[x]$ where K is a field.
  - But $K[x_1, \ldots, x_{n-1}]$ is only a ring.
  - Multivariate polynomials thus only support „pseudo-division“.

Computer Representation

- Prune mapping:
  - Represent only exponents/monomials with non-zero coefficients.
- Univariate polynomial representations:
  - Dense: coefficient sequence $[c_0, \dots, c_n]$
  - Sparse: exponent/coeff. sequence $[(e_0, c_0), \dots, (e_r, c_r)]$ with $e_i < e_{i+1}$.
- *n*-variate polynomial representations:
  - Recursive: univariate polynomial whose coefficients are $(n-1)$-variate polynomials (represented densely or sparsely).
  - Distributive: monomial/coefficient sequence $[(m_0, c_0), \dots, (m_r, c_r)]$ (typically represented sparsely).
    - Total order on monomials required for unique representation.
- Algorithmic efficiency:
  - Recursive algorithms based on isomorphism operate most efficiently with recursive representation.
  - Buchberger's Gröbner bases algorithm processes terms in any given „admissible“ order and profits from distributive rep. in that order.

General Approach

abstract type

representations	dense	sparse
recursive	✓	✓
distributive	✓	✓

General Approach

abstract type —— Map from monomials to coefficients

- Elegantly define basic operations
- Conveniently express algorithms, theorems, and proofs

representations	dense	sparse
recursive	✓	✓
distributive	✓	✓

General Approach

abstract type — Map from monomials to coefficients

- Elegantly define basic operations
- Conveniently express algorithms, theorems, and proofs

refinement ↓

- Theorems are preserved
- Representation values can instantiate variables of abstract type.

representations	dense	sparse
recursive	✓	✓
distributive	✓	✓

General Approach

abstract type — Map from monomials to coefficients

- Elegantly define basic operations
- Conveniently express algorithms, theorems, and proofs

refinement —

- Theorems are preserved
- Representation values can instantiate variables of abstract type.

representations	dense	sparse
recursive	✓	✓
distributive	✓	✓

$\xrightarrow{\text{generation}}$ executable code

General Approach

abstract type — Map from monomials to coefficients

- Elegantly define basic operations
- Conveniently express algorithms, theorems, and proofs

refinement —

- Theorems are preserved
- Representation values can instantiate variables of abstract type.

Every abstract algorithm is executable with every representation type.

representations	dense	sparse
recursive	✓	✓
distributive	✓	✓

$\xrightarrow{\text{generation}}$ executable code

Implementation in Isabelle

typedef $'a \Rightarrow_0 {}'b = \{ f :: {}'a \Rightarrow {}'b \mid \mathit{almost\text{-}everywhere\text{-}zero}\ f \}$

Implementation in Isabelle

typedef $'a \Rightarrow_0 'b = \{ f :: 'a \Rightarrow 'b \mid \mathit{almost\text{-}everywhere\text{-}zero}\ f \}$

$'a\ \mathit{mpoly}$

$'a\ \mathit{poly\text{-}rec}$

$'a\ \mathit{poly\text{-}distr}$

Implementation in Isabelle

typedef $'a \Rightarrow_0 \ 'b = \{ f :: \ 'a \Rightarrow 'b \mid \textit{almost-everywhere-zero}\ f \}$

$$
\begin{gathered}
(nat \Rightarrow_0 nat) \Rightarrow_0 \ 'a \\
\cong \\
'a\ mpoly
\end{gathered}
$$

$'a\ \textit{poly-rec}$ $\qquad$ $'a\ \textit{poly-distr}$

Implementation in Isabelle

typedef $'a \Rightarrow_0 {}'b = \{\, f :: {}'a \Rightarrow {}'b \mid \textit{almost-everywhere-zero}\ f \,\}$

$$\begin{array}{c} (\textit{nat} \Rightarrow_0 \textit{nat}) \Rightarrow_0 {}'a \\ \cong \\ {}'a\ \textit{mpoly} \end{array}$$

datatype $'a\ \textit{poly-rec}$
$\quad = \textit{Coeff}\ {}'a$
$\quad \mid \textit{Rec}\ (\textit{nat} \Rightarrow_0 {}'a\ \textit{poly-rec})$

$'a\ \textit{poly-distr}$

Implementation in Isabelle

typedef $'a \Rightarrow_0 {}'b = \{\, f :: {}'a \Rightarrow {}'b \mid \mathit{almost\text{-}everywhere\text{-}zero}\ f \}$

$$\begin{gathered}(\mathit{nat} \Rightarrow_0 \mathit{nat}) \Rightarrow_0 {}'a \\ \cong \\ {}'a\ \mathit{mpoly}\end{gathered}$$

datatype $'a\ \mathit{poly\text{-}rec}$
$\quad = \mathit{Coeff}\ {}'a$
$\quad \mid \mathit{Rec}\ (\mathit{nat} \Rightarrow_0 {}'a\ \mathit{poly\text{-}rec})$

$$\begin{gathered}{}'a\ \mathit{poly\text{-}distr} \\ \cong \\ {}'a\ \mathit{mpoly} \times \mathit{monom\text{-}order}\end{gathered}$$

Implementation in Isabelle

typedef $'a \Rightarrow_0 'b = \{ f :: 'a \Rightarrow 'b \mid \mathit{almost\text{-}everywhere\text{-}zero}\ f \}$

$$
\begin{array}{c}
(nat \Rightarrow_0 nat) \Rightarrow_0 'a \\
\cong \\
'a\ mpoly
\end{array}
$$

datatype $'a\ \mathit{poly\text{-}rec}$
$= \mathit{Coeff}\ 'a$
$\mid \mathit{Rec}\ (nat \Rightarrow_0 'a\ \mathit{poly\text{-}rec})$

$$
\begin{array}{c}
'a\ \mathit{poly\text{-}distr} \\
\cong \\
'a\ mpoly \times \mathit{monom\text{-}order} \\
\Big\downarrow \mathit{Rep} \\
((nat \Rightarrow_0 nat) \times 'a)\ list \times \mathit{monom\text{-}order}
\end{array}
$$

Implementation in Isabelle

typedef $'a \Rightarrow_0 'b = \{ f :: 'a \Rightarrow 'b \mid \mathit{almost\text{-}everywhere\text{-}zero}\ f \}$

$$\begin{tikzcd}
 & (\mathit{nat} \Rightarrow_0 \mathit{nat}) \Rightarrow_0 'a \arrow[d, phantom, "\cong"] & \\
 & 'a\ \mathit{mpoly} \arrow[dl, dashed] \arrow[dr, dashed] & \\
'a\ \mathit{poly\text{-}rec} \arrow[ur, bend left, "\mathit{Rec}"] \arrow[rr, dashed, leftrightarrow] & & 'a\ \mathit{poly\text{-}distr} \arrow[ul, bend right, "\mathit{Distr}"'] \arrow[d, phantom, "\cong"] \\
 & & 'a\ \mathit{mpoly} \times \mathit{monom\text{-}order} \arrow[d, "\mathit{Rep}"] \\
 & & ((\mathit{nat} \Rightarrow_0 \mathit{nat}) \times 'a)\ \mathit{list} \times \mathit{monom\text{-}order}
\end{tikzcd}$$

datatype $'a\ \mathit{poly\text{-}rec}$
$\quad = \mathit{Coeff}\ 'a$
$\quad \mid \mathit{Rec}\ (\mathit{nat} \Rightarrow_0 'a\ \mathit{poly\text{-}rec})$

Implementation in Isabelle

typedef $'a \Rightarrow_0 'b = \{ f :: 'a \Rightarrow 'b \mid \textit{almost-everywhere-zero}\ f \}$

$$(\textit{nat} \Rightarrow_0 \textit{nat}) \Rightarrow_0 'a \cong 'a\ \textit{mpoly}$$

$$\begin{tikzcd}
 & 'a\ \textit{mpoly} \arrow[dl, dashed] \arrow[dr, dashed] & \\
'a\ \textit{poly-rec} \arrow[ur, bend left, "\textit{Rec}"] \arrow[rr, dashed, leftrightarrow] & & 'a\ \textit{poly-distr} \arrow[ul, bend right, "\textit{Distr}"']
\end{tikzcd}$$

datatype $'a\ \textit{poly-rec}$
$= \textit{Coeff}\ 'a$
$\mid \textit{Rec}\ (\boxed{\textit{nat} \Rightarrow_0}\ 'a\ \textit{poly-rec})$

$'a\ \textit{poly-distr} \cong 'a\ \textit{mpoly} \times \textit{monom-order}$

$$'a\ \textit{mpoly} \times \textit{monom-order} \xrightarrow{\textit{Rep}} ((\boxed{\textit{nat} \Rightarrow_0}\ \textit{nat}) \times 'a)\ \textit{list} \times \textit{monom-order}$$

factor out dense &
sparse implementations

Design choice: number of variables

Should the number of variables show up in the type?

$'a$ *mpoly* vs. $'a$ *poly poly* ... *poly* vs. $('a, 7)$ *mpoly*

Design choice: number of variables

Should the number of variables show up in the type?

$'a\ mpoly$ vs. $'a\ poly\ poly \dots poly$ vs. $('a, 7)\ mpoly$

- Algorithms change number of variables dynamically.
- No computation on types

$$('a, 4 + 3)\ mpoly \neq ('a, 2 + 5)\ mpoly$$

Design choice: number of variables

Should the number of variables show up in the type?

$'a\ mpoly$ vs. $\cancel{'a\ poly\ poly \ldots poly}$ vs. $('a,\, 7)\ mpoly$

- Algorithms change number of variables dynamically.
- No computation on types

$$('a,\, 4+3)\ mpoly \neq ('a,\, 2+5)\ mpoly$$

Design choice: number of variables

Should the number of variables show up in the type?

$'a\ mpoly$ vs. $\cancel{'a\ poly\ poly \ldots poly}$ vs. $\cancel{('a, 7)\ mpoly}$

- Algorithms change number of variables dynamically.
- No computation on types

$$('a, 4 + 3)\ mpoly \neq ('a, 2 + 5)\ mpoly$$

Design choice: number of variables

Should the number of variables show up in the type?

$'a\ mpoly$ vs. $\cancel{'a\ poly\ poly \ldots poly}$ vs. $\cancel{('a, 7)\ mpoly}$

- Algorithms change number of variables dynamically.
- No computation on types

$$('a, 4 + 3)\ mpoly \neq ('a, 2 + 5)\ mpoly$$

- Polynomials over an unbounded number of variables

  Derived notion **variable number**:
  the highest index of a variable with non-zero coefficient.

  Implicitly extend polynomials as needed.

Exploit the Representation in Abstract Algorithms

Example:

- Gröbner bases algorithm depends on a **monomial order**.
- Efficiency relies on fast access to leading monomial in that order.

$$'a\ \mathit{mpoly} \qquad \text{vs.} \qquad 'a\ \mathit{mpoly} \times \mathit{monom\text{-}order}$$

Exploit the Representation in Abstract Algorithms

Example:

- Gröbner bases algorithm depends on a **monomial order**.
- Efficiency relies on fast access to leading monomial in that order.

$$'a\ \mathit{mpoly} \qquad \text{vs.} \qquad 'a\ \mathit{mpoly} \times \mathit{monom\text{-}order}$$

- Algebraic type classes require uniqueness of polynomials.

Exploit the Representation in Abstract Algorithms

Example:

- Gröbner bases algorithm depends on a **monomial order**.
- Efficiency relies on fast access to leading monomial in that order.

$$'a\ \mathit{mpoly} \qquad \text{vs.} \qquad \cancel{'a\ \mathit{mpoly} \times \mathit{monom\text{-}order}}$$

- Algebraic type classes require uniqueness of polynomials.

- Algorithm receives representational details as parameter.
- If polynomial's representation fits to the parameter, execution is **fast**.

  Otherwise, convert polynomial . . . or search . . .

- No static checks, no efficiency guarantees!

Open Problem: Controlling Representations

- What happens when we combine two polynomials?

+	*Rec*	Distr.
Rec	Rec	**???**
Distr	**???**	Distr

How can we make contextual information available?

Open Problem: Controlling Representations

- What happens when we combine two polynomials?

+	*Rec*	Distr.
Rec	Rec	???
Distr	???	Distr

How can we make contextual information available?

- How can the user specify the representations?

value *(2 :: int poly) * 3*

Recursive or distributive? Dense or Sparse? Which monomial order?

In CAS, the user declares his choice as a configuration option.
Can we mimick this in Isabelle?

The Ubiquitous Type Class *zero*

$\textbf{typedef}\ 'a \Rightarrow_0 'b = \{\, f :: 'a \Rightarrow 'b :: \mathit{zero} \mid \mathit{almost\text{-}everywhere\text{-}zero}\ f \,\}$

There is no map function for $'b$ that satisfies

$$\mathit{map}\ f \circ \mathit{map}\ g = \mathit{map}\ (f \circ g)$$

The Ubiquitous Type Class *zero*

typedef $'a \Rightarrow_0 {}'b = \{\, f :: {}'a \Rightarrow {}'b :: \mathit{zero} \mid \mathit{almost\text{-}everywhere\text{-}zero}\ f \,\}$

There is no map function for $'b$ that satisfies

$$\mathit{map}\ f \circ \mathit{map}\ g = \mathit{map}\ (f \circ g)$$

BNF	$\Rightarrow_0$ is not a BNF! Must construct $'a$ *poly-rec* manually
Lifting	Quotient theorem only for relations that respect *zero* No parametrised correspondence relations
Transfer	Transfer rules must restrict function space $===>$ is too weak
Library	Re-implement finite maps with the invariant $0 \notin \mathit{ran}\ m$ How can we improve reuse?

Summary

Current state of multivariate polynomials in Isabelle:

- ⊕ Design seems good
- ⊕ Prototype of abstract and representation types with minimal set of operations
- ⊖ Lemmas and algorithm implementations are still missing

Summary

Current state of multivariate polynomials in Isabelle:

- ⊕ Design seems good
- ⊕ Prototype of abstract and representation types with minimal set of operations
- ⊖ Lemmas and algorithm implementations are still missing

Up for discussion:

- User-friendliness/convenience for the working mathematician.
- Control of representations
- Better integration with Isabelle packages