Polynomials in Isabelle for the Working Mathematician

Walther Neuper, TU Graz
Wolfgang Schreiner, RISC Linz

Linz, Jun.2014

`hg clone https://hg.risc.uni-linz.ac.at/wneuper/poly`
Isabelle download `http://isabelle.in.tum.de` .

Slides `https://hg.risc.uni-linz.ac.at/wneuper/poly-demo.pdf`

Outline

1. Joint work Computer Algebra — Isabelle
2. Towards a Polynomial Package in Isabelle
   - An abstract polynomial for proofs
   - Proofs and algorithms within one system
   - Polynomial representations: distributive – recursive
   - Automated code generation preserves logical properties
3. Demo: the proof-of-concept prototype
   - Isabelle/jEdit: towards a prover IDE
   - Isabelle's mechanisms "typedef" and "instantiation"
   - Definitions – proofs – algorithms – code generation
   - Questions . . .
   - . . .

Outline

1. Joint work Computer Algebra — Isabelle
2. Towards a Polynomial Package in Isabelle
   - An abstract polynomial for proofs
   - Proofs and algorithms within one system
   - Polynomial representations: distributive – recursive
   - Automated code generation preserves logical properties
3. Demo: the proof-of-concept prototype
   - Isabelle/jEdit: towards a prover IDE
   - Isabelle's mechanisms "typedef" and "instantiation"
   - Definitions – proofs – algorithms – code generation
   - Questions . . .
   - . . .

Joint work

Aims	**Computer Algebra**	**Isabelle development**
For the working mathematician	*n.n.* *n.n.* design definitions prove theorems verify algorithms	
develop a polynomial package	*Wolfgang Schreiner* *RISC Linz* SAGE: basic operations representations proof: representations $\equiv$ proof: polynomial ring, . . .	*Andreas Lochbihler* *ETHZ Zürich* basic operations representations proof: representations $\equiv$ proof: polynomial ring, . . .
for efficient and verified algorithms.		*Florian Haftmann* *TU München* from Isabelle definitions generate code (Scala, Haskell, SML, . . .) preserving verification

Joint work

Aims	**Computer Algebra**	**Isabelle development**
For the working mathematician	*n.n.* *n.n.* design definitions prove theorems verify algorithms	
develop a polynomial package	*Wolfgang Schreiner* *RISC Linz* SAGE: basic operations representations proof: representations ≡ proof: polynomial ring, …	*Andreas Lochbihler* *ETHZ Zürich* basic operations representations proof: representations ≡ proof: polynomial ring, …
for efficient and verified algorithms.		*Florian Haftmann* *TU München* from Isabelle definitions generate code (Scala, Haskell, SML, …) preserving verification

Joint work

Aims	**Computer Algebra**	**Isabelle development**
For the working mathematician	*n.n.* *n.n.* design definitions prove theorems verify algorithms	
develop a polynomial package	*Wolfgang Schreiner* *RISC Linz* SAGE: basic operations representations proof: representations ≡ proof: polynomial ring, ...	*Andreas Lochbihler* *ETHZ Zürich* basic operations representations proof: representations ≡ proof: polynomial ring, ...
for efficient and verified algorithms.		*Florian Haftmann* *TU München* from Isabelle definitions generate code (Scala, Haskell, SML, ...) preserving verification

Joint work

Aims	**Computer Algebra**	**Isabelle development**
For the working mathematician	*n.n.* *n.n.* design definitions prove theorems verify algorithms	
develop a polynomial package	*Wolfgang Schreiner* *RISC Linz* SAGE: basic operations representations proof: representations ≡ proof: polynomial ring, . . .	*Andreas Lochbihler* *ETHZ Zürich* basic operations representations proof: representations ≡ proof: polynomial ring, . . .
for efficient and verified algorithms.		*Florian Haftmann* *TU München* from Isabelle definitions generate code (Scala, Haskell, SML, . . .) preserving verification

Joint work

Aims	**Computer Algebra**	**Isabelle development**
For the working mathematician	*n.n.* *n.n.* design definitions prove theorems verify algorithms	
develop a polynomial package	*Wolfgang Schreiner* *RISC Linz* SAGE: basic operations representations proof: representations ≡ proof: polynomial ring, . . .	*Andreas Lochbihler* *ETHZ Zürich* basic operations representations proof: representations ≡ proof: polynomial ring, . . .
for efficient and verified algorithms.		*Florian Haftmann* *TU München* from Isabelle definitions generate code (Scala, Haskell, SML, . . .) preserving verification

Joint work

Aims	**Computer Algebra**	**Isabelle development**
For the working mathematician	*n.n.* *n.n.* design definitions prove theorems verify algorithms	
develop a polynomial package	*Wolfgang Schreiner* *RISC Linz* SAGE: basic operations representations proof: representations $\equiv$ proof: polynomial ring, . . .	*Andreas Lochbihler* *ETHZ Zürich* basic operations representations proof: representations $\equiv$ proof: polynomial ring, . . .
for efficient and verified algorithms.		*Florian Haftmann* *TU München* from Isabelle definitions generate code (Scala, Haskell, SML, . . .) preserving verification

Joint work

Aims	**Computer Algebra**	**Isabelle development**
For the working mathematician	*n.n.* *n.n.* design definitions prove theorems verify algorithms	
develop a polynomial package	*Wolfgang Schreiner* *RISC Linz* SAGE: basic operations representations proof: representations $\equiv$ proof: polynomial ring, . . .	*Andreas Lochbihler* *ETHZ Zürich* basic operations representations proof: representations $\equiv$ proof: polynomial ring, . . .
for efficient and verified algorithms.		*Florian Haftmann* *TU München* from Isabelle definitions generate code (Scala, Haskell, SML, . . .) preserving verification

Joint work

Aims	**Computer Algebra**	**Isabelle development**
For the working mathematician	*n.n.* *n.n.* design definitions prove theorems verify algorithms	
develop a polynomial package	*Wolfgang Schreiner* *RISC Linz* SAGE: basic operations representations proof: representations $\equiv$ proof: polynomial ring, . . .	*Andreas Lochbihler* *ETHZ Zürich* basic operations representations proof: representations $\equiv$ proof: polynomial ring, . . .
for efficient and verified algorithms.		*Florian Haftmann* *TU München* from Isabelle definitions generate code (Scala, Haskell, SML, . . .) preserving verification

Outline

1. Joint work Computer Algebra — Isabelle
2. **Towards a Polynomial Package in Isabelle**
   - An abstract polynomial for proofs
   - Proofs and algorithms within one system
   - Polynomial representations: distributive – recursive
   - Automated code generation preserves logical properties
3. Demo: the proof-of-concept prototype
   - Isabelle/jEdit: towards a prover IDE
   - Isabelle's mechanisms “typedef” and “instantiation”
   - Definitions – proofs – algorithms – code generation
   - Questions . . .
   - . . .

Abstract polynomial

In Franz Winkler, Polynomial Algorithms, p.17:

Definition

An *n-variate polynomial over* the ring R is a mapping $p : \mathcal{N}_0^n \to R$, $(i_1, \ldots, i_n) \mapsto p_{1_i}, \ldots, p_{1_n}$ such that $p_{1_i}, \ldots, p_{1_n} = 0$ nearly everywhere. (Notation $p = \Sigma p_{1_i}, \ldots, p_{1_n} \cdot x^{i_1} \cdots x^{i_n}$)

In Isabelle's prototype[1]:

```
typedef ('a, 'b) poly_mapping =
  "{f :: 'a => 'b::zero. finite {x. f x ≠ 0}}"

typedef 'a mpoly =
  "UNIV::((nat,nat) poly_mapping, 'a::zero) poly_mapping
```

---

[1] https://hg.risc.uni-linz.ac.at/wneuper/poly/file/28e5ebbe5db5/Poly_Mapping.thy

Abstract polynomial

In Franz Winkler, Polynomial Algorithms, p.17:

Definition

An *n-variate polynomial over* the ring R is a mapping $p : \mathcal{N}_0^n \to R$, $(i_1, \ldots, i_n) \mapsto p_{1_i}, \ldots, p_{1_n}$ such that $p_{1_i}, \ldots, p_{1_n} = 0$ nearly everywhere. (Notation $p = \Sigma p_{1_i}, \ldots, p_{1_n} \cdot x^{i_1} \cdots x^{i_n}$)

In Isabelle's prototype[1]:

```
typedef ('a, 'b) poly_mapping =
  "{f ::  'a => 'b::zero.  finite {x.  f x ≠ 0}}"
```

```
typedef 'a mpoly =
  "UNIV::((nat,nat) poly_mapping, 'a::zero) poly_mapping
```

---

[1] https://hg.risc.uni-linz.ac.at/wneuper/poly/file/28e5ebbe5db5/Poly_Mapping.thy

Abstract polynomial

In Franz Winkler, Polynomial Algorithms, p.17:

Definition

An *n-variate polynomial over* the ring R is a mapping $p : \mathcal{N}_0^n \to R$, $(i_1, \ldots, i_n) \mapsto p_{1_i}, \ldots, p_{1_n}$ such that $p_{1_i}, \ldots, p_{1_n} = 0$ nearly everywhere. (Notation $p = \Sigma p_{1_i}, \ldots, p_{1_n} \cdot x^{i_1} \cdots x^{i_n}$)

In Isabelle's prototype[1]:

```
typedef ('a, 'b) poly_mapping =
  "{f ::  'a => 'b::zero.  finite {x.  f x ≠ 0}}"

typedef 'a mpoly =
  "UNIV::((nat,nat) poly_mapping, 'a::zero) poly_mapping
```

---

[1] https://hg.risc.uni-linz.ac.at/wneuper/poly/file/28e5ebbe5db5/Poly_Mapping.thy

Proofs and algorithms

A lemma and a (automated) proof

```
lemma gcd_mod:  "gcd (q * b + r) (b::int) = gcd b r"
  by (metis gcd_commute_int gcd_red_int
    mod_mult_self1 add_commute)
```

and an algorithm (written using Isabelle’s “function package”)

```
function euclid ::  "'a::ring_div => 'a => 'a"
  where "euclid a b =
    (if b = 0 then a else euclid b (a mod b))"
```

and a (omitted) proof about the algorithm

```
theorem euclid_gcd:  "euclid (a::ring_div) b = gcd a b"
  proof sorry
```

all within one system, within Isabelle.

Proofs and algorithms

A lemma and a (automated) proof

```
lemma gcd_mod:  "gcd (q * b + r) (b::int) = gcd b r"
  by (metis gcd_commute_int gcd_red_int
    mod_mult_self1 add_commute)
```

and an algorithm (written using Isabelle's "function package")

```
function euclid ::  "'a::ring_div => 'a => 'a"
  where "euclid a b =
    (if b = 0 then a else euclid b (a mod b))"
```

and a (omitted) proof about the algorithm

```
theorem euclid_gcd:  "euclid (a::ring_div) b = gcd a b"
  proof sorry
```

all within one system, within Isabelle.

Proofs and algorithms

A lemma and a (automated) proof

```
lemma gcd_mod:  "gcd (q * b + r) (b::int) = gcd b r"
  by (metis gcd_commute_int gcd_red_int
    mod_mult_self1 add_commute)
```

and an algorithm (written using Isabelle’s “function package”)

```
function euclid ::  "'a::ring_div => 'a => 'a"
  where "euclid a b =
    (if b = 0 then a else euclid b (a mod b))"
```

and a (omitted) proof about the algorithm

```
theorem euclid_gcd:  "euclid (a::ring_div) b = gcd a b"
  proof sorry
```

all within one system, within Isabelle.

Proofs and algorithms

A lemma and a (automated) proof

```
lemma gcd_mod:  "gcd (q ⋆ b + r) (b::int) = gcd b r"
  by (metis gcd_commute_int gcd_red_int
    mod_mult_self1 add_commute)
```

and an algorithm (written using Isabelle’s “function package”)

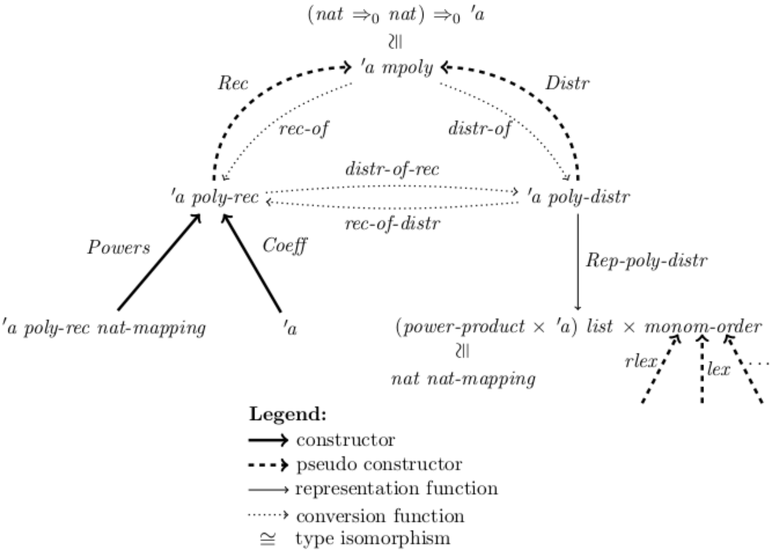
```
function euclid ::  "'a::ring_div => 'a => 'a"
  where "euclid a b =
    (if b = 0 then a else euclid b (a mod b))"
```

and a (omitted) proof about the algorithm

```
theorem euclid_gcd:  "euclid (a::ring_div) b = gcd a b"
  proof sorry
```

all within one system, within Isabelle.

Polynomial representations



$(nat \Rightarrow_0 nat) \Rightarrow_0 {'a}$

$\cong$

'a mpoly

Rec

Distr

rec-of

distr-of

distr-of-rec

'a poly-rec

'a poly-distr

rec-of-distr

Powers

Coeff

Rep-poly-distr

'a poly-rec nat-mapping

'a

(power-product × 'a) list × monom-order

$\cong$

nat nat-mapping

rlex

lex

. . .

Legend:

$\longrightarrow$ constructor

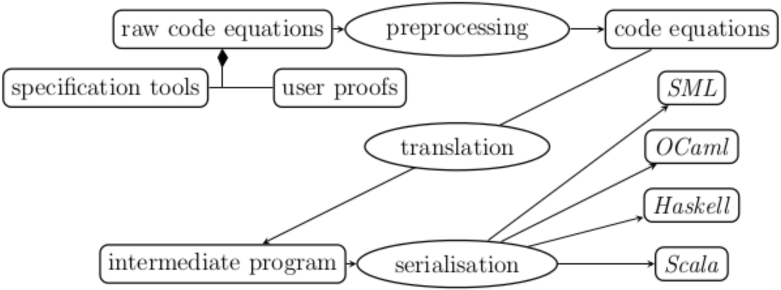
- - -> pseudo constructor

$\longrightarrow$ representation function

······> conversion function

$\cong$ type isomorphism

Automated code generation



raw code equations

preprocessing

code equations

specification tools

user proofs

translation

intermediate program

serialisation

SML

OCaml

Haskell

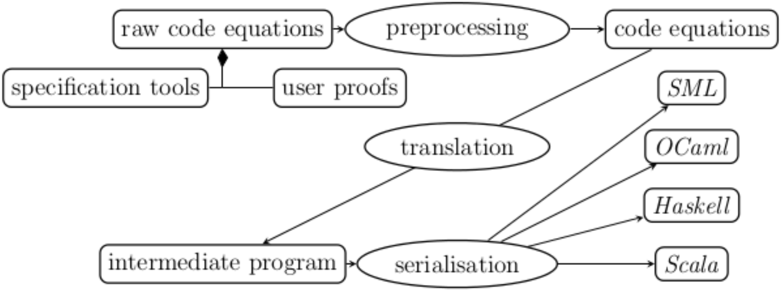
Scala

"code equations" are equational theorems in Isabelle/HOL.

Partial correctness of theorems transfers to generated programs: rewrite steps in the program can be simulated in the logic.

The generator's components can be customized individually.

Automated code generation

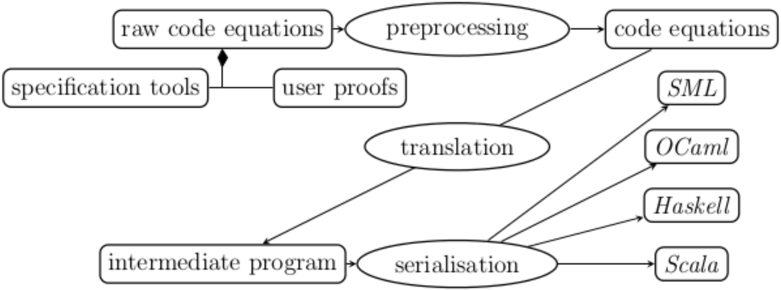


"code equations" are equational theorems in Isabelle/HOL.

Partial correctness of theorems transfers to generated programs: rewrite steps in the program can be simulated in the logic.

The generator's components can be customized individually.

Automated code generation



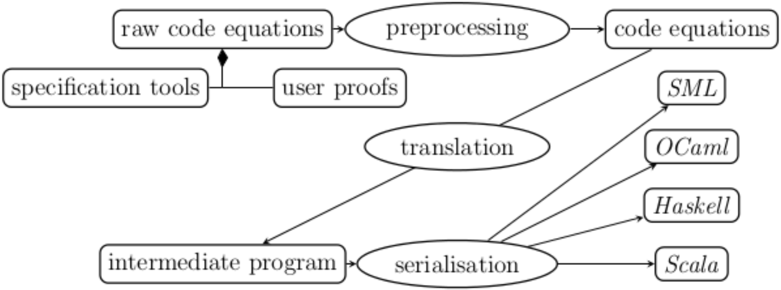
raw code equations
preprocessing
code equations
specification tools
user proofs
translation
intermediate program
serialisation
SML
OCaml
Haskell
Scala

"code equations" are equational theorems in Isabelle/HOL.

Partial correctness of theorems transfers to generated programs: rewrite steps in the program can be simulated in the logic.

The generator's components can be customized individually.

Automated code generation



"code equations" are equational theorems in Isabelle/HOL.

Partial correctness of theorems transfers to generated programs: rewrite steps in the program can be simulated in the logic.

The generator's components can be customized individually.

Outline

1. Joint work Computer Algebra — Isabelle
2. Towards a Polynomial Package in Isabelle
   - An abstract polynomial for proofs
   - Proofs and algorithms within one system
   - Polynomial representations: distributive – recursive
   - Automated code generation preserves logical properties
3. Demo: the proof-of-concept prototype
   - Isabelle/jEdit: towards a prover IDE
   - Isabelle's mechanisms "typedef" and "instantiation"
   - Definitions – proofs – algorithms – code generation
   - Questions . . .
   - . . .

Isabelle/jEdit

see Poly_Demo.thy

“typedef” and “instantiation”

see Poly_Demo.thy

Definitions – proofs – . . .

see Poly_Demo.thy

Questions

see Poly_Demo.thy

Thank you for attention !

`hg clone https://hg.risc.uni-linz.ac.at/wneuper/poly`
Isabelle download `http://isabelle.in.tum.de`

Short introduction to the prototype package:

`https://hg.risc.uni-linz.ac.at/wneuper/polyintro.pdf`