ICMS 2016

Common Divisors of Solvable Polynomials in JAS

Heinz Kredel, University of Mannheim

ICMS 2016, Berlin

Overview

- Introduction
  - Example
- Solvable Polynomial Rings
  - Parametric Solvable Polynomial Rings
- Generic Common Divisors
  - Recursive Algorithm
  - Class Design
  - Example cont.
- Conclusions

Introductory example

- solvable polynomial ring
  - variables x, y, z, t, relations in Q_x
  - Residue class quotient field modulo the two-sided ideal I

$$\mathbb{Q}(x, y, z, t; Q_x)_{/\mathcal{I}}\{r; Q_r\}$$

$$Q_x = \{z * y = yz + x, t * y = yt + y, t * z = zt - z\} \quad Q_r = \emptyset$$

$$\mathcal{I} = (t^2 + z^2 + y^2 + x^2 + 1)$$

as Java code

Ruby and Python interface also available

- Polynomial ring construction

```
String[] vars = new String[] { "x", "y", "z", "t" };
BigRational cfac = new BigRational(1);
GenSolvablePolynomialRing<BigRational> mfac;
mfac = new GenSolvablePolynomialRing<BigRational>(cfac,
                         TermOrderByName.INVLEX, vars);
GenSolvablePolynomial<BigRational> p;
List<GenSolvablePolynomial<BigRational>> rel;
rel = new ArrayList<GenSolvablePolynomial<BigRational>>();

//z, y,  y * z + x,
p = mfac.parse("z"); rel.add(p);
p = mfac.parse("y"); rel.add(p);
p = mfac.parse("y * z + x"); rel.add(p);
//t, y,  y * t + y, //t, z,  z * t - z  ...
mfac.addSolvRelations(rel);          //add relations to Q_x
```

Java code

- Local residue ring construction

```
p = mfac.parse("t^2 + z^2 + y^2 + x^2 + 1"); F.add(p);
SolvableIdeal<BigRational> id;
id = new SolvableIdeal<BigRational>(mfac, F,
                                    SolvableIdeal.Side.twosided);
id.doGB(); // compute twosided GB
SolvableLocalResidueRing<BigRational> efac;
efac = new SolvableLocalResidueRing<BigRational>(id);
```

Expression swell

```
SolvableLocalResidue<BigRational> a, b, c, d, e, f;
p = mfac.parse("t + x + y + 1");
a = new SolvableLocalResidue<BigRational>(efac, p);
p = mfac.parse("z^2+x+1");
b = new SolvableLocalResidue<BigRational>(efac, p);
c = a.inverse();
f = b.multiply(c).multiply(a);
b.equals(f); // --> true, since (b * 1/a) * a == b
```

- b equals $(b*a^{-1})*a = f$
- but b has 4 terms and f has 150 terms
- need some kind of reduction to lower terms
- like division by common divisors in the commutative case

Related work (selected)

- enveloping fields of Lie algebras [Apel, Lassner]
- solvable polynomial rings [Kandri-Rodi, Weispfenning]
- free-noncommutative polynomial rings [Mora]
- parametric solvable polynomial rings and comprehensive Gröbner bases [Weispfenning, Kredel]
- recursive solvable polynomial rings [Kredel]
- PBW algebras in Singular/Plural [Levandovskyy]
- primary ideal decomposition [Gomez-Torrecillas]

Solvable Polynomial Rings

Solvable polynomial ring S: associative Ring (S,0,1,+,-,*), K a (skew) field, in n variables

$$S = \mathbf{K}\{X_1, \ldots, X_n; Q; Q'\}$$

commutator relations between variables, $lt(p_{ij}) < X_i X_j$

$$Q = \{X_j * X_i = c_{ij} X_i X_j + p_{ij} : 0 \neq c_{ij} \in \mathbf{K}, X_i X_j > p_{ij} \in S, 1 \leq i < j \leq n\}$$

commutator relations between variables and coefficients

$$Q' = \{X_i * a = c_{ai} a X_i + p_{ai} : 0 \neq c_{ai} \in \mathbf{K}, p_{ai} \in \mathbf{K}, 1 \leq i \leq n, a \in \mathbf{K}\}$$

< a *-compatible term order on S x S: a < b ⇒ a*c < b*c and c*a < c*b for a, b, c in S

Parametric Solvable Polynomial Coefficient Rings

$$S = \mathbf{R}\{U_1, \ldots, U_m; Q_u\}\{X_1, \ldots, X_n; Q_x; Q'_{ux}\}$$

$$Q_u = \{\ U_j * U_i = c_{uij} U_i U_j + p_{uij} : \\ 0 \neq c_{uij} \in \mathbf{R}, U_i U_j > p_{uij} \in R, 1 \leq i < j \leq m\ \}$$

$$Q_x = \{\ X_j * X_i = c_{xij} X_i X_j + p_{xij} : \\ 0 \neq c_{xij} \in R, X_i X_j > p_{xij} \in S, 1 \leq i < j \leq n\ \}$$

$$Q'_{ux} = \{\ X_j * U_i = c_{ij} U_i X_j + p_{ij} : \\ 0 \neq c_{ij} \in \mathbf{R}, U_i X_j > p_{ij} \in S, 1 \leq i \leq m, 1 \leq j \leq n\ \}$$

recursive solvable polynomial rings

$$S_k = \mathbf{R}\{X_1, \ldots, X_k; Q_k\}\{X_{k+1}, \ldots, X_n; Q_n; Q'_{kn}\}, \quad 0 \leq k \leq n$$

Factorization

- solvable polynomial rings are integral domains and factorization domains (if coefficients are so)
- but factorization may not be uniqe
- obviously the order of the factors matters
- we have to distiguish between left and right (common) divisors
- implementation is work in progress

Ore condition

- for a, b in R there exist
  - c, d in R with c*a = d*b   left Ore condition
  - c', d' in R with a*c' = b*d'   right Ore condition
- Theorem: Noetherian rings satify the Ore condition
  - left / left and right / right
- can be computed by left respectively right syzygy computations in R [Apel]
- Theorem: domains with Ore condition can be embedded in a skew field
- a/b * c/d :=: (f*c)/(e*b) where e,f with e*a = f*d

Implementation of Solvable Polynomial Rings

- Java Algebra System (JAS)
- generic type parameters : RingElem<C>
- type safe, interoperable, object oriented
- has commutative greatest common divisors, squarefree decomposition, factorization and Gröbner bases
- scriptable with JRuby, Jython and interactive
- parallel multi-core and distributed cluster algorithms

Implementation (cont.)

- solvable polynomials can share representations with commutative polynomials and reuse implementations, "only" multiplication to be done
- implementation is generic in the sense that various coefficient rings can be used in a strongly type safe way and still good performing code
- employ parametric coefficient rings with commutator relations between variables and coefficient variables
- special case recursive solvable polynomial rings

Recursive solvable polynomial ring

- implemented in `RecSolvablePolynomial` and `RecSolvablePolynomialRing`
- extends `GenSolvablePolynomial<GenPolynomial<C>>`
- additional relation table `coeffTable` for relations from Q'_{ux}, with type `RelationTable<GenPolynomial<C>>`
- recording of powers of relations for lookup instead of recomputation
- new method `rightRecursivePolynomial()` with coefficients on the right side

Common divisor algorithm

- for a univariate polynomial compute (with multivariate coefficients) compute the (left, right) content by recursion
- remove both contents to obtain a primitive polynomial
- for the univariate polynomial compute a common divisor with Euclids algorithm and pseudo remainders
- note: pseudo remainders need the computaton of Ore conditions

Idea of recursive algorithm

```
uPol uGcd( uPol a, uPol b ) {      // Euclids algorithm
    while ( b != 0 ) {
          // let a = q b + r;           // (left) remainder
          // let ldcf(b)^e a = q b + r; // pseudo remainder
          a = b;
          b = r; // simplify remainder
    }
    return a;
}
```

```
mPol uGcd( mPol a, mPol b ) {
     a1 = content(a);    // gcd of coefficients
     b1 = content(b);    // or recursion
     c1 = gcd( a1, b1 ); // recursion
     a2 = a / a1;        // primitive part
     b2 = b / b1;
     c2 = uGcd( a2, b2 );
     return c1 * c2;
}
```

ICMS 2016

Common Divisors

ICMS 2016

Class design

- interface `GreatestCommonDivisor`
  - with leftGcd() and rightGcd()
  - with leftContent() and rightContent()
  - with leftPrimitivePart() and rightPrimitivePart()
  - construct lists of mutable common divisors 1
  - type parameter C: extends interface GcdRingElem

Class design (cont.)

- abstract class `GreatestCommonDivisorsAbstract`
  - implements all methods from interface except
  - leftBaseGcd(), rightBaseGcd(), leftRecursiveUnivariateGcd(), rightRecursiveUnivariateGcd()
- implemented by concrete classes `GreatestCommonDivisorSimple` and `GreatestCommonDivisorPrimitive`
  - using the respective polynomial remainder sequences (PRS)

Extensions

- `SGCDFactory` with getImplementation() or get Proxy()
- `SGCDParallelProxy` using invokeAny() of ExecutorService in java.util.concurrent
  - run two algorithms in parallel and use result of first finished one

Example (continued)

```
GreatestCommonDivisorAbstract<BigRational> engine;
engine = new GreatestCommonDivisorSimple<BigRational>(cfac);
p = engine.leftGcd(f.num,f.den);
```

```
// p = ( x**2 * z * t**2 + 3 * x * z * t**2 + 2 * z * t**2 + x**2 *
// t**2 + 3 * x * t**2 + 2 * t**2 + z**2 * t + 2 * x**2 * y * z * t + 6
// * x * y * z * t + 4 * y * z * t + 2 * x**3 * z * t + 9 * x**2 * z * t
// + 14 * x * z * t + 8 * z * t + 2 * x**2 * y * t + 6 * x * y * t + 4 *
// y * t + 5 * x**3 * t + 19 * x**2 * t + 23 * x * t + 9 * t + y * z**2
// + x * z**2 + 3 * z**2 + x**2 * y**2 * z + 3 * x * y**2 * z + 2 * y**2
// * z + 2 * x**3 * y * z + 10 * x**2 * y * z + 17 * x * y * z + 10 * y
// * z + x**4 * z + 6 * x**3 * z + 12 * x**2 * z + 15 * x * z + 6 * z +
// x**2 * y**2 + 3 * x * y**2 + 2 * y**2 + 5 * x**3 * y + 20 * x**2 * y
// + 26 * x * y + 11 * y + 4 * x**4 + 19 * x**3 + 36 * x**2 + 31 * x + 7)
```

```
GenSolvablePolynomial<BigRational>[] qr;
qr = FDUtil.<BigRational> rightBasePseudoQuotientRemainder(f.num,
fn = qr[0]; // ( z**2 + x + 1 ), qr[1] == 0
qr = FDUtil.<BigRational> rightBasePseudoQuotientRemainder(f.den,
fd = qr[0]; // 1, qr[1] == 0
e = new SolvableLocalResidue<BigRational>(efac, fn, fd);
// e = ( z**2 + x + 1 )
e.equals(b); // --> true
```

Application

- make use of the common divisor computation in the constructors of `SolvableLocalResidue`, `SolvableLocal` and `SolvableQuotient`
- so reduce the fractions to lower terms
- utility methods leftGcdCofactors and rightGcdCofactors
- improve the feasibility of computations with solvable quotient rings
- expression swell in Ore condition remains
    - syzygy computation in rings with k variables

Conclusions

- considered parametric solvable polynomial rings, with definition of commutator relations between polynomial variables and coefficient variables
- computed in recursive solvable polynomial rings
- possible to compute common divisors on the left and right side
- use to simplify quotients of solvable polynomials
- using them as coefficient rings of solvable polynomial rings makes computations of roots and ideal constructions over skew fields feasible

Conclusions (cont.)

- algorithms implemented in JAS in a type-safe, object oriented way with generic coefficients
- the high complexity of the solvable multiplication
- presented an efficient simplifier to reduce (intermediate) expression swell to foster practical computations
- high complexity of Ore condition computations remain (syzygies for multivariate polynomials)
- this will eventually be improved in future work

Thank you for your attention

Questions ?

Comments ?

http://krum.rz.uni-mannheim.de/jas/

Acknowledgments

- thanks to: Thomas Becker, Raphael Jolly, Wolfgang K. Seiler, Axel Kramer, Thomas Sturm, Victor Levandovskyy, Joachim Apel, Hans-Günther Kruse, Markus Aleksy
- thanks to the referees