# Gröbner Bases and Applications

# in Java

Heinz Kredel

Computer-Algebra Seminar FSS 2008
University of Mannheim

# Overview

- Coefficients and Polynomials
  - Types and Classes
  - Functionality and Implementation
  - Examples and Performance
- Polynomial Reduction
- Gröbner Bases and Buchberger algorithm
- Applications
  - Ideals
  - Residue class rings

# Introduction to software

- object oriented design of a computer algebra system

  = software collection for symbolic (non-numeric) computations
- type safe through Java generic types
- thread safe, ready for multi-core CPUs
- dynamic memory system with GC
- 64-bit ready
- jython (Java Python) front end

# Polynomials

$$p \in R = C[x_1, \ldots, x_n]$$

$$p = 3x_1^2 x_3^4 + 7x_2^5 - 61 \in \mathbb{Z}[x_1, x_2, x_3]$$

- multivariate polynomials
- polynomial ring
  - in n variables
  - over a coefficient ring
- 3 variables $x_1, x_2 \text{ and } x_3$
- with integer coefficients

# Monoid rings

$T$ *Monoid*, $C$ *Ring*

$$p: T \to C$$
$$t \to p(t) = c_t$$

$p(t) \neq 0$ *for only finitely many* $t \in T$

$$(p+q)(t) = p(t) + q(t)$$

$$(p \cdot q)(t) = \sum_{uv=t} p(u) \cdot q(v)$$

$$T = \{ x_1^{e_1} \ldots x_n^{e_n} \mid (e_1, \ldots, e_n) \in \mathbb{N}^n \}$$

$$C[x_{1,\ldots}, x_n] = \{ p \mid p: T \to C \}$$

- polynomials as mappings
  - from terms to coefficients
  - terms are power products of variables
  - with finite support
  - definition of sum and product

$x_1^2 x_3^4 \to 3, \quad x_2^5 \to 7, \quad x_1^0 x_2^0 x_3^0 \to -61$

*else* $\quad x_1^{e_1} x_2^{e_2} x_3^{e_3} \to 0$

# Polynomials (cont.)

$$one: \left\{ x_1^0 x_2^0 \ldots x_n^0 \rightarrow 1 \right\}$$

$$zero: \left\{ \right\}$$

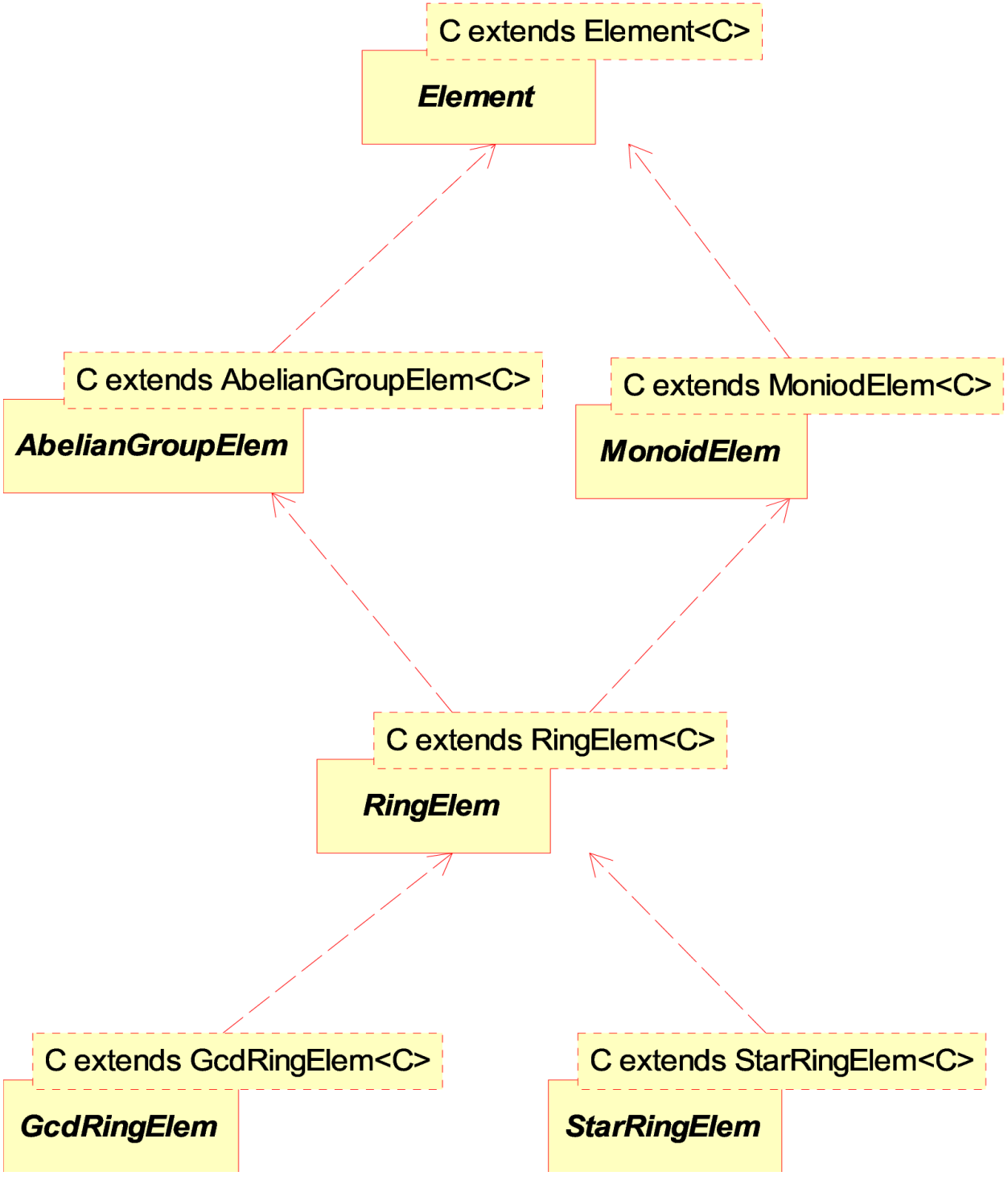
$$x_1^2 x_3^4 \; >_T \; x_2^5$$

- mappings to zero are not stored
- terms are ordered / sorted

$$x_j * x_i = c_{ij} x_i x_j + p_{ij}$$

$$1 \leqslant i < j \leqslant n, \;\; 0 \neq c_{ij} \in C,$$
$$x_i x_j >_T p_{ij} \in R$$

- polynomials with non-commutative multiplication
- commutative is special case $c_{ij}=1,\ p_{ij}=0$


C extends Element<C>
Element
C extends AbelianGroupElem<C>
AbelianGroupElem
C extends MoniodElem<C>
MonoidElem
C extends RingElem<C>
RingElem
C extends GcdRingElem<C>
GcdRingElem
C extends StarRingElem<C>
StarRingElem

# Ring element creation

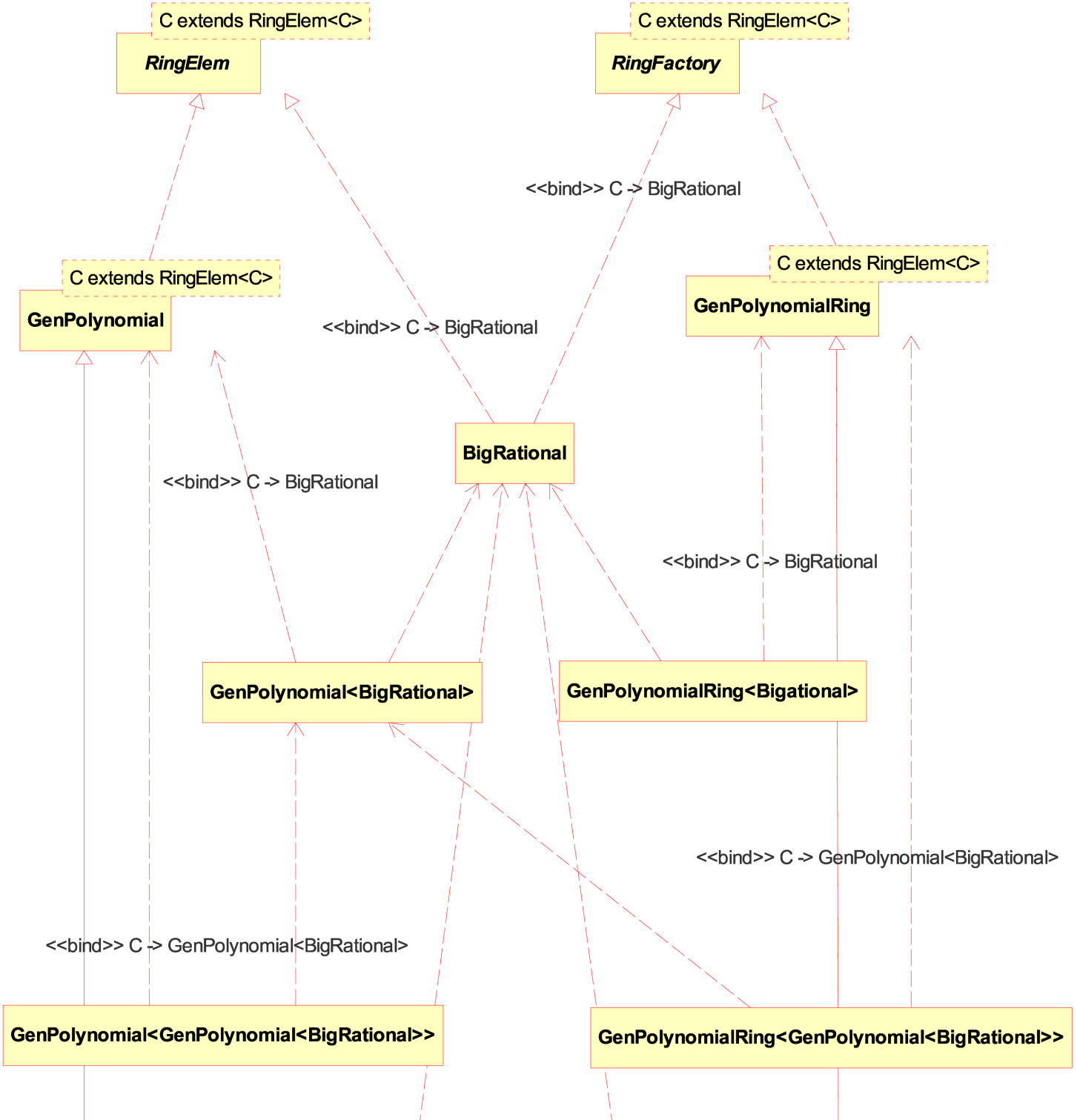
- recursive type for coefficients and polynomials
- creation of ZERO and ONE needs information about the ring
- `new C()` not allowed in Java, `C` type parameter
- solution with factory pattern: `RingFactory`
- factory has sufficient information for creation of ring elements
- eventually has references to other factories, e.g. for coefficients

# Ring element functionality

- `C` is type parameter
- `C sum(C S), C subtract(C S), C negate(), C abs()`
- `C multiply(C s), C divide(C s), C remainder(C s), C inverse()`
- `boolean isZERO(), isONE(), isUnit(), int signum()`
- `equals(Object b), int hashCode(), int compareTo(C b)`
- `C clone()` versus `C copy(C a)`
- `Serializable` interface is implemented

# Ring factory functionality

- create 0 and 1
  - `C getZERO(), C getONE()`
- `C copy(C a)`
- embed integers `C fromInteger(long a)`
  - `C fromInteger(java.math.BigInteger a)`
- random elements `C random(int n)`
- parse string representations
  - `C parse(String s), C parse(Reader r)`
- `isCommutative(), isAssociative()`


C extends RingElem<C>
RingElem
C extends RingElem<C>
RingFactory
<<bind>> C -> BigRational
C extends RingElem<C>
GenPolynomial
<<bind>> C -> BigRational
C extends RingElem<C>
GenPolynomialRing
BigRational
<<bind>> C -> BigRational
<<bind>> C -> BigRational
GenPolynomial<BigRational>
GenPolynomialRing<Bigational>
<<bind>> C -> GenPolynomial<BigRational>
<<bind>> C -> GenPolynomial<BigRational>
GenPolynomial<GenPolynomial<BigRational>>
GenPolynomialRing<GenPolynomial<BigRational>>

# Coefficients

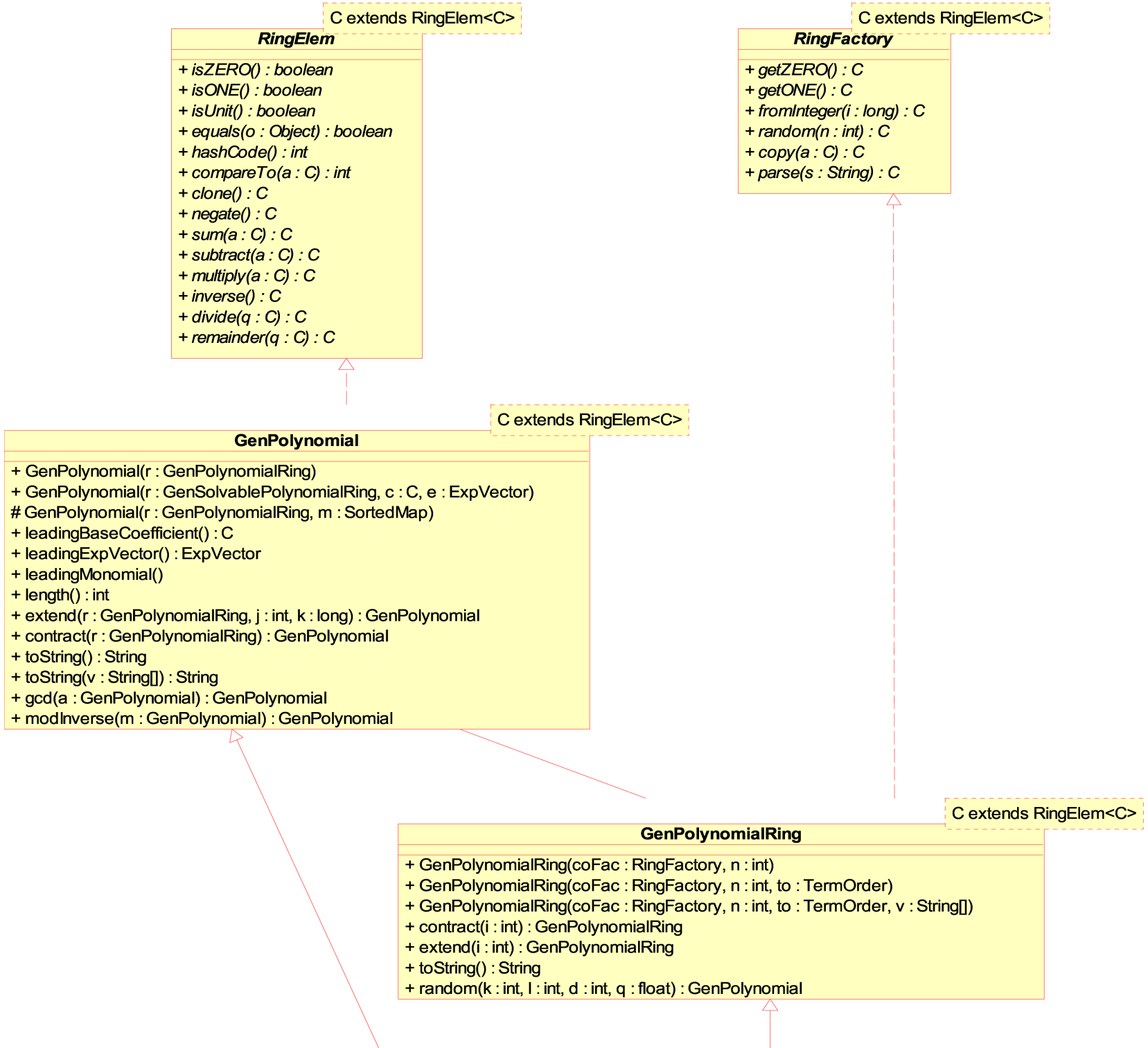
- e.g. `BigRational, BigInteger`
- implement both interfaces
- creation of rational number 2 from long 2:
  - `new BigRational(2)`
  - `cfac.fromInteger(2)`
- creation of rational number 1/2 from two longs:
  - `new BigRational(1,2)`
  - `cfac.parse(  1/2  )`

# Polynomials

- `GenPolynomial<C extends RingElem<C>>`
- `C` is coefficient type in the following
- implements `RingElem<GenPolynomial<C>>`
- factory is `GenPolynomialRing<...>`
- implements `RingFactory<GenPolynomial<C>>`
- factory constructors require coefficient factory parameter

# Polynomial creation

- types are
  - `GenPolynomial<BigRational>`
  - `GenPolynomialRing<BigRational>`
- creation is
  - `new GenPolynomialRing<BigRational>(cfac,5)`
  - `pfac.getONE()`
  - `pfac.parse(  1  )`
- polynomials as coefficients
  - `GenPolynomial<GenPolynomial<BigRational>>`
  - `GenPolynomialRing<GenPolynomial<...>>(pfac,3)`


C extends RingElem<C>
RingElem
+ isZERO() : boolean
+ isONE() : boolean
+ isUnit() : boolean
+ equals(o : Object) : boolean
+ hashCode() : int
+ compareTo(a : C) : int
+ clone() : C
+ negate() : C
+ sum(a : C) : C
+ subtract(a : C) : C
+ multiply(a : C) : C
+ inverse() : C
+ divide(q : C) : C
+ remainder(q : C) : C
C extends RingElem<C>
RingFactory
+ getZERO() : C
+ getONE() : C
+ fromInteger(i : long) : C
+ random(n : int) : C
+ copy(a : C) : C
+ parse(s : String) : C
C extends RingElem<C>
GenPolynomial
+ GenPolynomial(r : GenPolynomialRing)
+ GenPolynomial(r : GenSolvablePolynomialRing, c : C, e : ExpVector)
# GenPolynomial(r : GenPolynomialRing, m : SortedMap)
+ leadingBaseCoefficient() : C
+ leadingExpVector() : ExpVector
+ leadingMonomial()
+ length() : int
+ extend(r : GenPolynomialRing, j : int, k : long) : GenPolynomial
+ contract(r : GenPolynomialRing) : GenPolynomial
+ toString() : String
+ toString(v : String[]) : String
+ gcd(a : GenPolynomial) : GenPolynomial
+ modInverse(m : GenPolynomial) : GenPolynomial
C extends RingElem<C>
GenPolynomialRing
+ GenPolynomialRing(coFac : RingFactory, n : int)
+ GenPolynomialRing(coFac : RingFactory, n : int, to : TermOrder)
+ GenPolynomialRing(coFac : RingFactory, n : int, to : TermOrder, v : String[])
+ contract(i : int) : GenPolynomialRing
+ extend(i : int) : GenPolynomialRing
+ toString() : String
+ random(k : int, l : int, d : int, q : float) : GenPolynomial

# Polynomial factory constructors

- coefficient factory of the corresponding type
- number of variables
- term order (optional)
- names of the variables (optional)
- `GenPolynomialRing<C>( RingFactory<C> cf, int n, TermOrder t, String[] v)`

$$x_1^2 x_3^4 \; >_T \; x_2^5$$

# Polynomial factory functionality

- ring factory methods plus more specific methods
- `GenPolynomial<C> random(int k, int l, int d, float q, Random rnd)`
- embed and restrict polynomial ring to ring with more or less variables
  - `GenPolynomialRing<C> extend(int i)`
  - `GenPolynomialRing<C> contract(int i)`
  - `GenPolynomialRing<C> reverse()`
- handle term order adjustments

# Polynomial functionality

- ring element methods plus more specific methods
- constructors all require a polynomial factory
  - `GenPolynomial(GenPolynomialRing<C> r, C c, ExpVector e)`
  - `GenPolynomial(GenPolynomialRing<C> r, SortedMap<ExpVector,C> v)`
- access parts of polynomials
  - `ExpVector leadingExpVector()`
  - `C leadingBaseCoefficient()`
  - `Map.Entry<ExpVector,C> leadingMonomial()`
- extend and contract polynomials

# Example

```
BigInteger z = new BigInteger();
TermOrder to = new TermOrder();
String[] vars = new String[] { "x1", "x2", "x3" };
GenPolynomialRing<BigInteger> ring

 = new GenPolynomialRing<BigInteger>(z,3,to,vars);

GenPolynomial<BigInteger> pol
 = ring.parse( "3 x1^2 x3^4 + 7 x2^5 - 61" );


toString output:
ring = BigInteger(x1, x2, x3) IGRLEX
pol = GenPolynomial[
      3 (4,0,2), 7 (0,5,0), -61 (0,0,0) ]
pol = 3 x1^2 * x3^4 + 7 x2^5 - 61
```

# Example (cont.)

```
p1 = pol.subtract(pol);
p2 = pol.multiply(pol);
```

```
p1 = GenPolynomial[  ]
p1 = 0
p2 =  9 x1^4 * x3^8 + 42 x1^2 * x2^5 * x3^4
     + 49 x2^10
     - 366 x1^2 * x3^4 - 854 x2^5 + 3721
```

# Chebychev polynomials

defined by recursion:

```
T[0] = 1
T[1] = x
T[n] = 2 x T[n-1] - T[n-2]
```

first 10 polynomials:

```
T[0] = 1
T[1] = x
T[2] = 2 x^2 - 1
T[3] = 4 x^3 - 3 x
T[4] = 8 x^4 - 8 x^2 + 1
T[5] = 16 x^5 - 20 x^3 + 5 x
T[6] = 32 x^6 - 48 x^4 + 18 x^2 - 1
T[7] = 64 x^7 - 112 x^5 + 56 x^3 - 7 x
T[8] = 128 x^8 - 256 x^6 + 160 x^4 - 32 x^2 + 1
T[9] = 256 x^9 - 576 x^7 + 432 x^5 - 120 x^3 + 9 x
```

# Chebychev polynomial computation

```
 1.  int m = 10;
 2.  BigInteger fac = new BigInteger();
 3.  String[] var   = new String[]{ "x" };
 4.  GenPolynomialRing<BigInteger> ring
 5.                 = new GenPolynomialRing<BigInteger>(fac,1,var);
 6.  List<GenPolynomial<BigInteger>> T
 7.                 = new ArrayList<GenPolynomial<BigInteger>>(m);
 8.  GenPolynomial<BigInteger> t, one, x, x2;
 9.  one = ring.getONE();
10.  x   = ring.univariate(0); // polynomial in variable 0
11.  x2  = ring.parse("2 x");
12.  T.add( one );    // T[0]
13.  T.add( x );      // T[1]
14.  for ( int n = 2; n < m; n++ ) {
15.      t = x2.multiply( T.get(n-1) ).subtract( T.get(n-2) );
16.      T.add( t );   // T[n]
17.  }
18.  for ( int n = 0; n < m; n++ ) {
19.      System.out.println("T["+n+"] = " + T.get(n) );
20.  }
```

# Implementation

- 160 classes and interfaces
- plus 80 JUnit test cases
- JDK 1.5 with generic types
- logging with Apache Log4j
- some jython scripts
- javadoc API documentation
- revision control with subversion
- build tool is Apache Ant
- open source, license is GPL

# Coefficient implementation

- `BigInteger` based on `java.math.BigInteger`
- implemented in pure Java, no GMP C-library
- using adaptor pattern to implement `RingElem` (and `RingFactory`) interface
- about 10 to 15 times faster than the Modula-2 implementation `SACI` (in 2000)
- other classes: `BigRational`, `ModInteger`, `BigComplex`, `BigQuaternion` and `BigOctonion`
- `AlgebraicNumber` class can be used over `BigRational` or `ModInteger`

# Polynomial implementation

- are (ordered) maps from terms to coefficients
- implemented with `SortedMap` interface and `TreeMap` class from Java collections framework
- alternative implementation with `Map` and `LinkedHashMap`, which preserves the insertion order
- but had inferior performance
- terms (the keys) are implemented by class `ExpVector`
- coefficients implement `RingElem` interface

# Polynomial implementation (cont.)

- `ExpVector` is dense array of exponents (as `long`) of variables
- sparse array, array of `int, Long` not implemented
- would like to have `ExpVector<long>`
- polynomials are intended as immutable objects
- object variables are `final` and the map is not modified after creation
- eventually wrap with `unmodifiableSortedMap()`
- avoids synchronization in multi threaded code

# Performance

- polynomial arithmetic performance:
  - performance of coefficient arithmetic
    - `java.math.BigInteger` in pure Java, faster than GMP style JNI C version
  - sorted map implementation
    - from Java collection classes with known efficient algorithms
  - exponent vector implementation
    - using `long[]`, have to consider also `int[]` or `short[]`
    - want `ExpVector<C>` but generic types may not be elementary types
  - JAS comparable to general purpose CA systems but slower than specialized systems

# Performance

$$compute\ q = p \times (p+1)$$

$$p = (1+x+y+z)^{20}$$
$$p = (10000000001\,(1+x+y+z))^{20}$$
$$p = (1+x^{2147483647}+y^{2147483647}+z^{2147483647})^{20}$$

| JAS: options, system | JDK 1.5 | JDK 1.6 |
|---|---|---|
| BigInteger, G | 16.2 | 13.5 |
| BigInteger, L | 12.9 | 10.8 |
| BigRational, L, s | 9.9 | 9.0 |
| BigInteger, L, s | 9.2 | **8.4** |
| BigInteger, L, big e, s | 9.2 | **8.4** |
| BigInteger, L, big c | 66.0 | 59.8 |
| BigInteger, L, big c, s | 45.0 | **43.2** |

| options, system | time | @2.7GHz |
|---|---|---|
| MAS 1.00a, L, GC = 3.9 | **33.2** | |
| Singular 2-0-6, G | 2.5 | |
| Singular, L | **2.2** | |
| Singular, G, big c | 12.9 | |
| Singular, L, big exp | out of memory | |
| Maple 9.5 | **15.2** | 9.1 |
| Maple 9.5, big e | 19.8 | 11.8 |
| Maple 9.5, big c | 64.0 | 38.0 |
| Mathematica 5.2 | **22.8** | 13.6 |
| Mathematica 5.2, big e | 30.9 | 18.4 |
| Mathematica 5.2, big c | 30.6 | 18.2 |
| JAS, s | 8.4 | 5.0 |
| JAS, big e, s | 8.6 | 5.1 |
| JAS, big c, s | 47.8 | 28.5 |

Computing times in seconds on AMD 1.6 GHz or 2.7 GHz Intel XEON CPU.
Options are: coefficient type, term order: G = graded, L = lexicographic,
big c = using the big coefficients, big e = using the big exponents, s = server JVM.

# Overview

- Coefficients and Polynomials
  - Types and Classes
  - Functionality and Implementation
  - Examples and Performance
- Polynomial Reduction
- Gröbner Bases and Buchberger algorithm
- Applications
  - Ideals
  - Residue class rings

# Term orderings

- (M,<) monoid with linear order <
- < admissible for M:

$$1_M \leq u, \text{ for all } u \in M$$

$$u < v \text{ implies } uw < vw, \text{ for all } u, v, w \in M$$

- (inverse) lexicographical orders
- graded orders
- block orders
- orders defined by weight vectors and matrices
- orders defined by real univariate polynomials

# Term order implementation

- need comparators for `SortedMap<ExpVector,C>(Comparator<ExpVector>)`
- generated from class `TermOrder(INVLEX)`
- has methods
  - `Comparator<ExpVector> getDescendComparator()`
  - `Comparator<ExpVector> getAscendComparator()`
- implemented all practically used orders
  - (inverse) lexicographical
  - (inverse) graded, i.e. total degree
  - defined by weight matrices
  - elimination orders (split or block orders)

# Reduction

- removal of terms in a polynomial by subtraction of multiples of a polynomial taken from a list of polynomials
- extended reduction

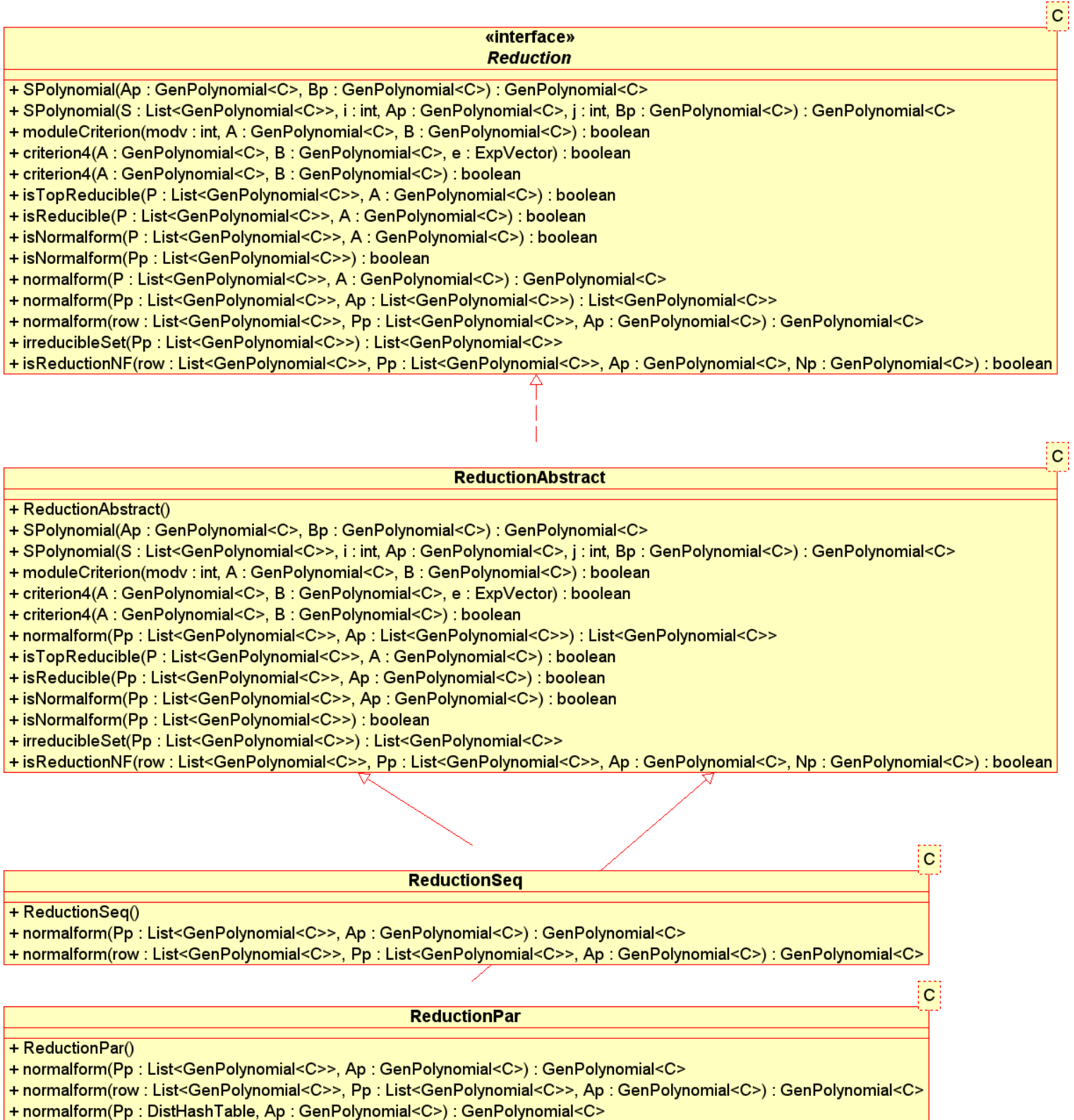
$$R = K[x_1, \ldots, x_n],\ f, g, h \in R,\ F \subset R,\ a \in K,\ t \in T$$

$$f \rightarrow_g f',\quad f' = f - a\,t\,g,\ LM(a\,t\,g) = m \in f$$

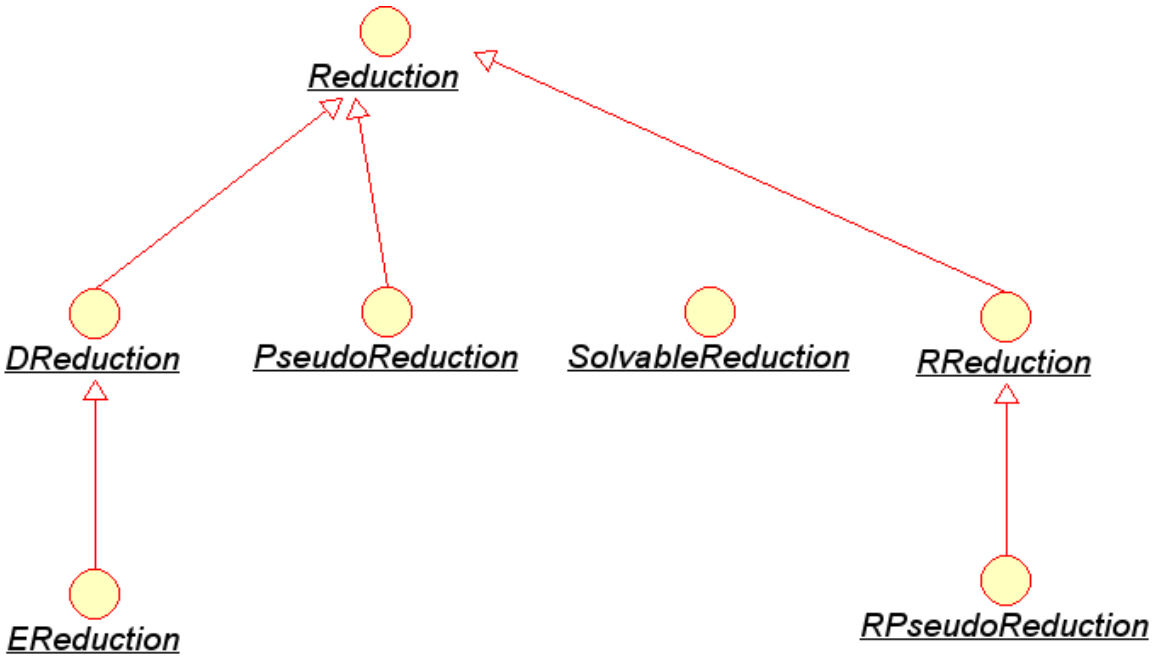
$$f \rightarrow_F h,\quad f = f_0 \rightarrow_{g_1} f_1 \rightarrow \ldots \rightarrow_{g_k} f_k = h,\ g_i \in F$$

$$Lemma: f' <_T f,\ h <_T f,\ k > 0$$

$$f \rightarrow_F h,\quad f = \sum_{g \in F} h_g g + h,\quad for\ some\ h_g \in R,\ g \in F$$


C
«interface»
Reduction
+ SPolynomial(Ap : GenPolynomial<C>, Bp : GenPolynomial<C>) : GenPolynomial<C>
+ SPolynomial(S : List<GenPolynomial<C>>, i : int, Ap : GenPolynomial<C>, j : int, Bp : GenPolynomial<C>) : GenPolynomial<C>
+ moduleCriterion(modv : int, A : GenPolynomial<C>, B : GenPolynomial<C>) : boolean
+ criterion4(A : GenPolynomial<C>, B : GenPolynomial<C>, e : ExpVector) : boolean
+ criterion4(A : GenPolynomial<C>, B : GenPolynomial<C>) : boolean
+ isTopReducible(P : List<GenPolynomial<C>>, A : GenPolynomial<C>) : boolean
+ isReducible(P : List<GenPolynomial<C>>, A : GenPolynomial<C>) : boolean
+ isNormalform(P : List<GenPolynomial<C>>, A : GenPolynomial<C>) : boolean
+ isNormalform(Pp : List<GenPolynomial<C>>) : boolean
+ normalform(P : List<GenPolynomial<C>>, A : GenPolynomial<C>) : GenPolynomial<C>
+ normalform(Pp : List<GenPolynomial<C>>, Ap : List<GenPolynomial<C>>) : List<GenPolynomial<C>>
+ normalform(row : List<GenPolynomial<C>>, Pp : List<GenPolynomial<C>>, Ap : GenPolynomial<C>) : GenPolynomial<C>
+ irreducibleSet(Pp : List<GenPolynomial<C>>) : List<GenPolynomial<C>>
+ isReductionNF(row : List<GenPolynomial<C>>, Pp : List<GenPolynomial<C>>, Ap : GenPolynomial<C>, Np : GenPolynomial<C>) : boolean
C
ReductionAbstract
+ ReductionAbstract()
+ SPolynomial(Ap : GenPolynomial<C>, Bp : GenPolynomial<C>) : GenPolynomial<C>
+ SPolynomial(S : List<GenPolynomial<C>>, i : int, Ap : GenPolynomial<C>, j : int, Bp : GenPolynomial<C>) : GenPolynomial<C>
+ moduleCriterion(modv : int, A : GenPolynomial<C>, B : GenPolynomial<C>) : boolean
+ criterion4(A : GenPolynomial<C>, B : GenPolynomial<C>, e : ExpVector) : boolean
+ criterion4(A : GenPolynomial<C>, B : GenPolynomial<C>) : boolean
+ normalform(Pp : List<GenPolynomial<C>>, Ap : List<GenPolynomial<C>>) : List<GenPolynomial<C>>
+ isTopReducible(P : List<GenPolynomial<C>>, A : GenPolynomial<C>) : boolean
+ isReducible(Pp : List<GenPolynomial<C>>, Ap : GenPolynomial<C>) : boolean
+ isNormalform(Pp : List<GenPolynomial<C>>, Ap : GenPolynomial<C>) : boolean
+ isNormalform(Pp : List<GenPolynomial<C>>) : boolean
+ irreducibleSet(Pp : List<GenPolynomial<C>>) : List<GenPolynomial<C>>
+ isReductionNF(row : List<GenPolynomial<C>>, Pp : List<GenPolynomial<C>>, Ap : GenPolynomial<C>, Np : GenPolynomial<C>) : boolean
C
ReductionSeq
+ ReductionSeq()
+ normalform(Pp : List<GenPolynomial<C>>, Ap : GenPolynomial<C>) : GenPolynomial<C>
+ normalform(row : List<GenPolynomial<C>>, Pp : List<GenPolynomial<C>>, Ap : GenPolynomial<C>) : GenPolynomial<C>
C
ReductionPar
+ ReductionPar()
+ normalform(Pp : List<GenPolynomial<C>>, Ap : GenPolynomial<C>) : GenPolynomial<C>
+ normalform(row : List<GenPolynomial<C>>, Pp : List<GenPolynomial<C>>, Ap : GenPolynomial<C>) : GenPolynomial<C>
+ normalform(Pp : DistHashTable, Ap : GenPolynomial<C>) : GenPolynomial<C>

# Implemented reductions




: DReductionSeq

: ReductionAbstract

: SolvableReductionAbstract

: EReductionSeq

: ReductionSeq

: SolvableReductionSeq

: ReductionPar

: RReductionSeq

: PseudoReductionSeq

: RPseudoReductionSeq

# Overview

- Coefficients and Polynomials
  - Types and Classes
  - Functionality and Implementation
  - Examples and Performance
- Polynomial Reduction
- Gröbner Bases and Buchberger algorithm
- Applications
  - Ideals
  - Residue class rings

# Gröbner Bases

$$K \text{ field}, R = K[x_1, \ldots, x_n], f, g \in R, G \subset R, a, b \in K, s, t \in T$$

$$\text{Definition}: G \text{ is Gröbner base iff } id(LM(G)) = LM(id(G))$$

$$\text{Theorem}: G \text{ is Gröbner base iff } \forall_{f \in id(G)} f \rightarrow_G 0$$

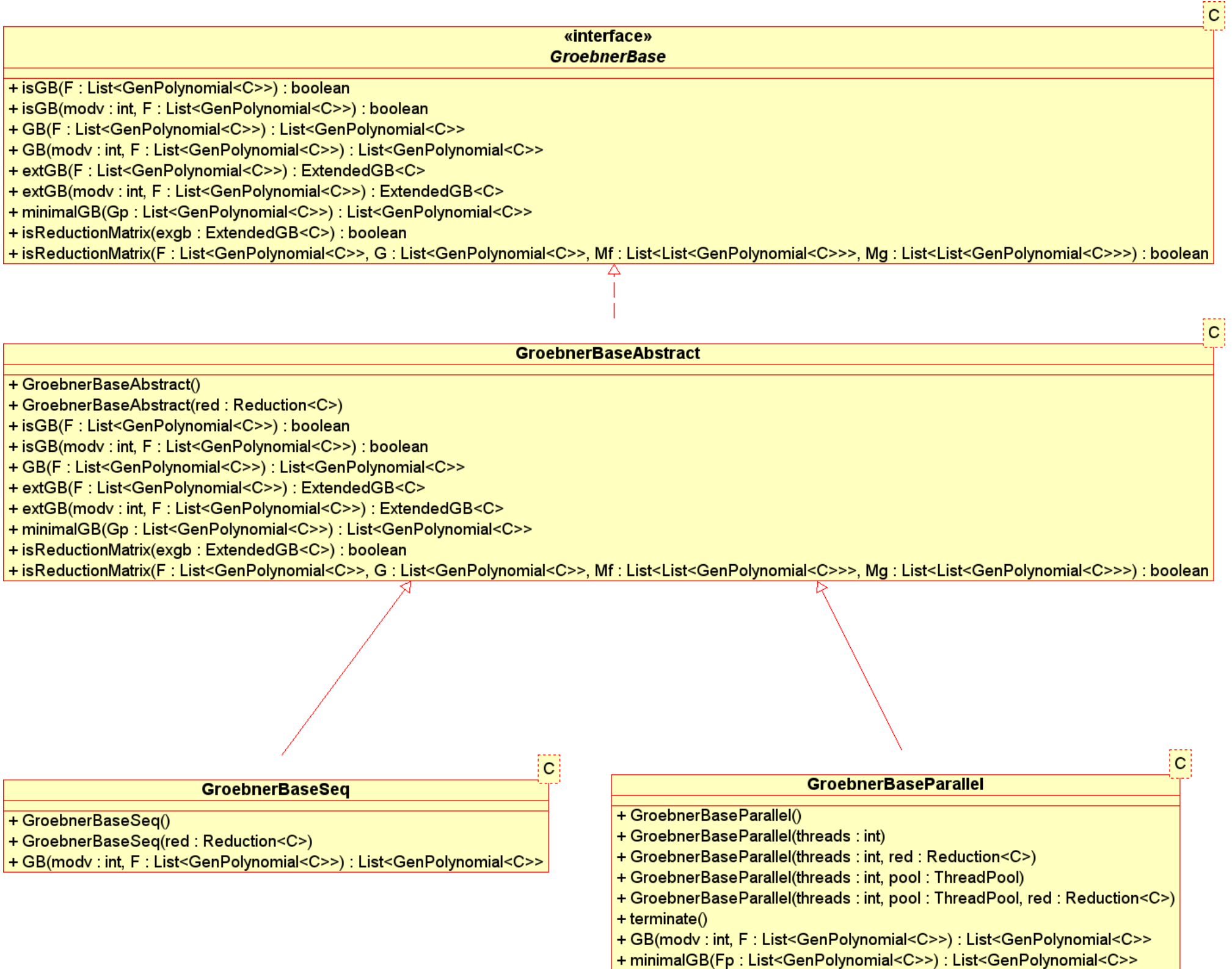
$$\text{Definition}: Spol(f, g) = a\,t\,f - b\,s\,g, \text{ with } LM(a\,t\,f) = LM(b\,s\,g)$$

$$\text{Theorem}: G \text{ is Gröbner base iff } \forall_{f, g \in G, f \neq g} Spol(f, g) \rightarrow_G 0$$

# Buchberger algorithm

```
algorithm: G = GB( F )
input: F a list of polynomials in R
output: G a Gröbner Base of ideal(F)

G = F
B = { (f,g) | f, g in G, f != g }
while B != {} do
  select and remove (f,g) from B
  s = Spol(f,g)
  h = normalform(G,s)
  if h != 0 then
     for f in G add (f,h) to B
     add h to G
  end
end
return G
```


«interface»
GroebnerBase
+ isGB(F : List<GenPolynomial<C>>) : boolean
+ isGB(modv : int, F : List<GenPolynomial<C>>) : boolean
+ GB(F : List<GenPolynomial<C>>) : List<GenPolynomial<C>>
+ GB(modv : int, F : List<GenPolynomial<C>>) : List<GenPolynomial<C>>
+ extGB(F : List<GenPolynomial<C>>) : ExtendedGB<C>
+ extGB(modv : int, F : List<GenPolynomial<C>>) : ExtendedGB<C>
+ minimalGB(Gp : List<GenPolynomial<C>>) : List<GenPolynomial<C>>
+ isReductionMatrix(exgb : ExtendedGB<C>) : boolean
+ isReductionMatrix(F : List<GenPolynomial<C>>, G : List<GenPolynomial<C>>, Mf : List<List<GenPolynomial<C>>>, Mg : List<List<GenPolynomial<C>>>) : boolean
C
GroebnerBaseAbstract
+ GroebnerBaseAbstract()
+ GroebnerBaseAbstract(red : Reduction<C>)
+ isGB(F : List<GenPolynomial<C>>) : boolean
+ isGB(modv : int, F : List<GenPolynomial<C>>) : boolean
+ GB(F : List<GenPolynomial<C>>) : List<GenPolynomial<C>>
+ extGB(F : List<GenPolynomial<C>>) : ExtendedGB<C>
+ extGB(modv : int, F : List<GenPolynomial<C>>) : ExtendedGB<C>
+ minimalGB(Gp : List<GenPolynomial<C>>) : List<GenPolynomial<C>>
+ isReductionMatrix(exgb : ExtendedGB<C>) : boolean
+ isReductionMatrix(F : List<GenPolynomial<C>>, G : List<GenPolynomial<C>>, Mf : List<List<GenPolynomial<C>>>, Mg : List<List<GenPolynomial<C>>>) : boolean
C
GroebnerBaseSeq
+ GroebnerBaseSeq()
+ GroebnerBaseSeq(red : Reduction<C>)
+ GB(modv : int, F : List<GenPolynomial<C>>) : List<GenPolynomial<C>>
C
GroebnerBaseParallel
+ GroebnerBaseParallel()
+ GroebnerBaseParallel(threads : int)
+ GroebnerBaseParallel(threads : int, red : Reduction<C>)
+ GroebnerBaseParallel(threads : int, pool : ThreadPool)
+ GroebnerBaseParallel(threads : int, pool : ThreadPool, red : Reduction<C>)
+ terminate()
+ GB(modv : int, F : List<GenPolynomial<C>>) : List<GenPolynomial<C>>
+ minimalGB(Fp : List<GenPolynomial<C>>) : List<GenPolynomial<C>>

# Improved Buchberger algorithm

- avoid reductions to zero if possible
- Lemma $\mathrm{LT}(f)\mathrm{LT}(g) = lcm(\mathrm{LT}(f), \mathrm{LT}(g))$ *implies* $Spol(f, g) \to_{\{f,g\}} 0$
- Proposition *if* exists $p$ *with* $\mathrm{LT}(p) \mid lcm(\mathrm{LT}(f), \mathrm{LT}(g))$ and $Spol(f, p) \to 0$ and $Spol(p, g) \to 0$ *then* $Spol(f, g) \to 0$
- Gebauer-Möller
- Faugere: F4, F5
- Faugere, Gianni, Lazard, Mora: change of ordering

# Trinks example

```
String exam = "Rat(B,S,T,Z,P,W) L "
            + "( "
            + "( 45 P + 35 S - 165 B - 36 ), "
            + "( 35 P + 40 Z + 25 T - 27 S ), "
            + "( 15 W + 25 S P + 30 Z - 18 T - 165 B**2 ), "
            + "( - 9 W + 15 T P + 20 S Z ), "
            + "( P W + 2 T Z - 11 B**3 ), "
            + "( 99 W - 11 B S + 3 B**2 ), "
            + "( B**2 + 33/50 B + 2673/10000 ) "
            + ") ";
Reader source = new StringReader( exam );
GenPolynomialTokenizer parser = new GenPolynomialTokenizer( source );
PolynomialList<BigRational> F;
try {
    F = (PolynomialList<BigRational>) parser.nextPolynomialSet();
} catch(ClassCastException e) {
} catch(IOException e) {
}
GroebnerBase<BigRational> bb = new GroebnerBaseSeq<BigRational>();
G = bb.GB(F.list);
assertTrue("isGB( GB(Trinks7) )", bb.isGB(G) );
PolynomialList<BigRational> t = new PolynomialList<BigRational>(F.ring,G
System.out.println("G = " + t);
```

# Trinks example in jython

```
r = Ring( "Rat(B,S,T,Z,P,W) L" );
print "Ring: " + str(r); print;
ps = """
(
 ( 45 P + 35 S - 165 B - 36 ),
 ( 35 P + 40 Z + 25 T - 27 S ),
 ( 15 W + 25 S P + 30 Z - 18 T - 165 B**2 ),
 ( - 9 W + 15 T P + 20 S Z ),
 ( P W + 2 T Z - 11 B**3 ),
 ( 99 W - 11 B S + 3 B**2 ),
 ( 10000 B**2 + 6600 B + 2673 )
)
""";
f = r.ideal( ps ); print "Ideal: " + str(f); print; #startLog();

rg = f.GB(); print "seq Output:", rg; print;
rg = f.parGB(2);
f.distClient(); # starts in background
rg = f.distGB(2);

terminate(); sys.exit();
```

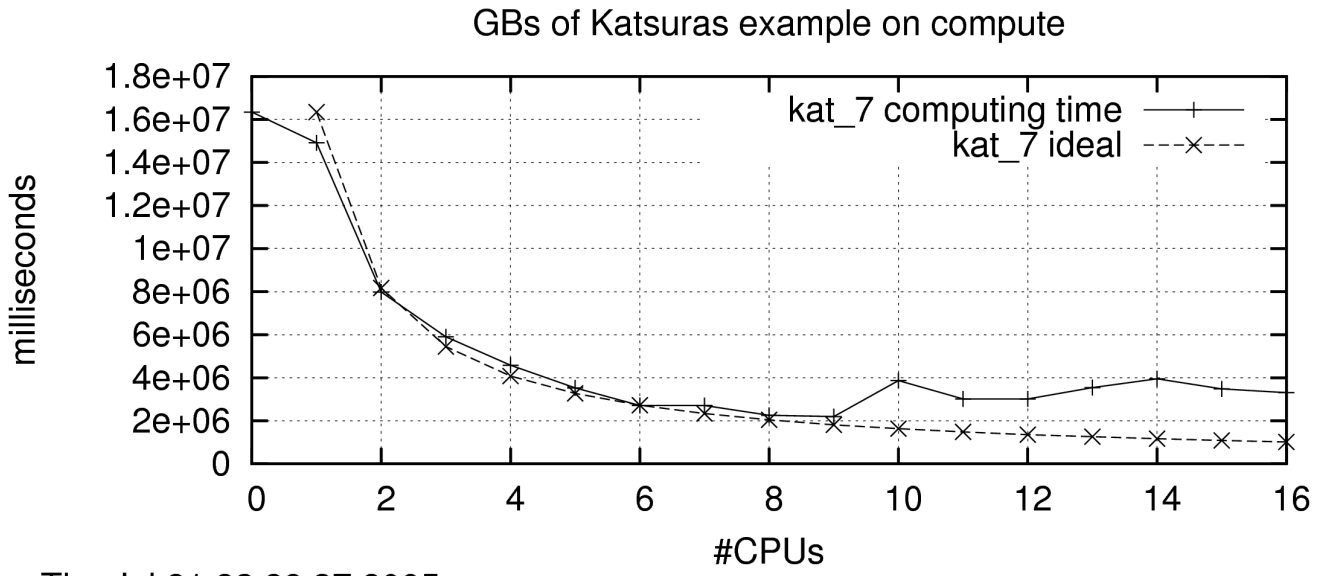
# Trinks example output

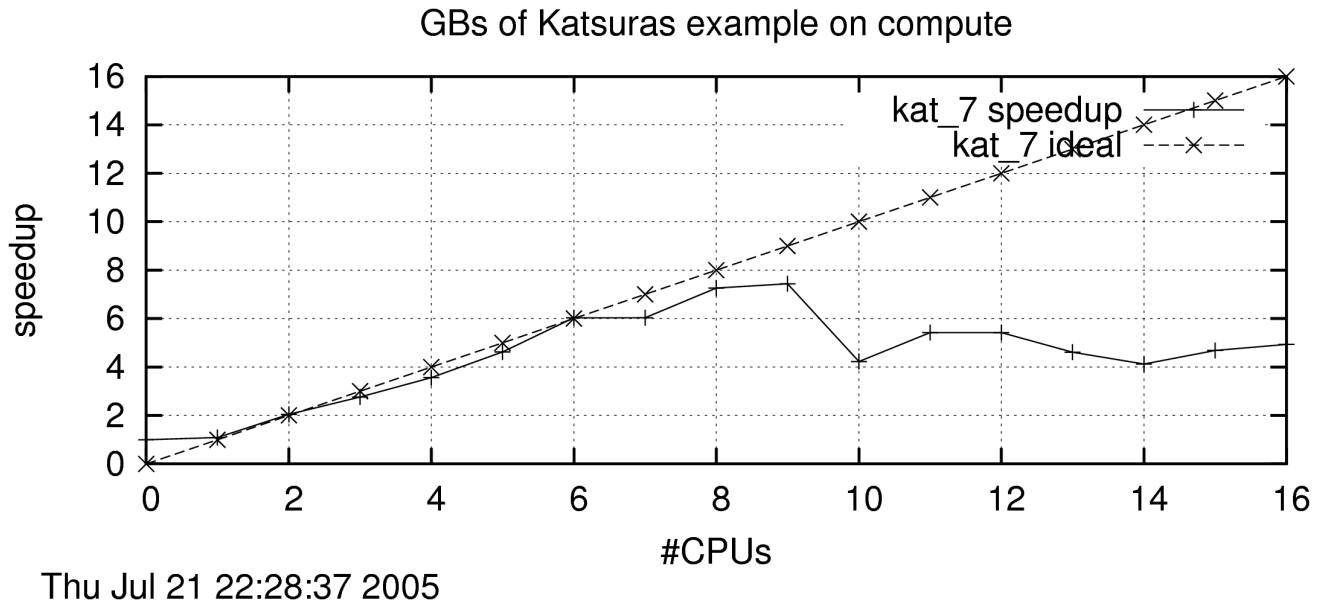
```
Ideal: BigRational( B, S, T, Z, P, W ) INVLEX
(
( 10000 B^2 + 6600 B + 2673  ),
( 45 P + 35 S - 165 B - 36  ),
( 35 P + 40 Z + 25 T - 27 S ),
( 15 W + 25 S * P + 30 Z - 18 T - 165 B^2 ),
( -9 W + 15 T * P + 20 S * Z ),
( 99 W - 11 B * S + 3 B^2 ),
( P * W + 2 T * Z - 11 B^3 )
)

sequential executed in 131 ms
seq Output: BigRational( B, S, T, Z, P, W ) INVLEX
(
( B^2 + 33/50 B + 2673/10000  ),
( S - 5/2 B - 9/200  ),
( T - 37/15 B + 27/250  ),
( Z + 49/36 B + 1143/2000  ),
( P - 31/18 B - 153/200  ),
( W + 19/120 B + 1323/20000  )
)
parallel-new 2 executed in 184 ms
distributed 2 executed in 1138 ms (131 ms start-up)
```

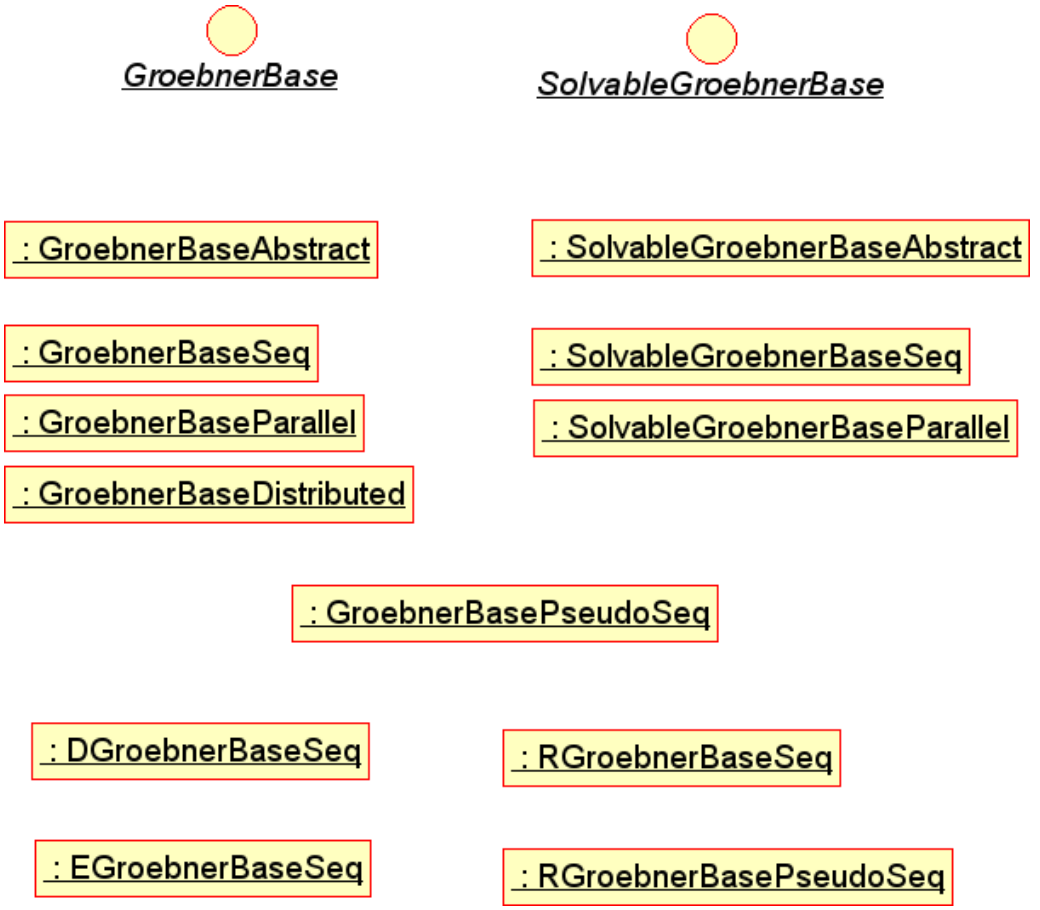
# Parallel Gröbner bases

- work queue of polynomials `CriticalPairList`
- with synchronized methods `get(), put(), removeNext()` to modify data structure
- scales well for 8 CPUs on a well structured problem (see next figures)
- distributed version uses some kind of a distributed list to store polynomials of set (implemented by a DHT)
- use of object serialization for transport of polynomials over the network


GBs of Katsuras example on compute
kat_7 computing time
kat_7 ideal
milliseconds
#CPUs
1.8e+07
1.6e+07
1.4e+07
1.2e+07
1e+07
8e+06
6e+06
4e+06
2e+06
0
0
2
4
6
8
10
12
14
16
Thu Jul 21 22:28:37 2005


GBs of Katsuras example on compute
kat_7 speedup
kat_7 ideal
speedup
#CPUs
16
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14
16
Thu Jul 21 22:28:37 2005

# Implemented Gröbner Bases



# Applications (1)

- ideal membership test
  - proper ideal test
- elimination ideals
- intersection of ideals

$$R=K[x_1,\dots,x_n],\quad F, F_{1,} F_2 \subset R,\quad f\in R$$

$$f\in id(F) \;\Leftrightarrow\; f\to_G 0,\; G=GB(F)$$

$$id(F)=R \;\Leftrightarrow\; 1\in id(F) \;\Leftrightarrow\; 1\in G,\; G=GB(F)$$

$$X=\{x_1,\dots,x_n\},\quad Y=\{x_{i_1},\dots,x_{i_k}\}\subset X,\quad Z=X\setminus Y,\quad Y\ll Z \quad block\ order$$

$$J=id(F)\cap K[Y] \;\Leftrightarrow\; J=id(G\cap K[Y]),\; G=GB(F)\;\; wrt\;\; Y\ll Z$$

$$J=id(F_1)\cap id(F_2)\Leftrightarrow J=id(G\cap K[X]), G=GB(t F_1+(1-t)F_2), X\ll t$$

# Applications (2)

- extended Gröbner base
- syzygies

$G = GB(F),\ \ want\ \ \hat{G} = \{q_{gf}\},\ \ \hat{F} = \{p_{fq}\},\ \ q_{gf}, p_{fg} \in R\ \ with$

$g = \sum_{f \in F} q_{gf} f \ \text{ and } \ f = \sum_{g \in G} p_{fg} g \ \ for \ \ g \in G, f \in F$

$algorithm: \ (G, \hat{G}, \hat{F}) \ = \ extGB(F)$

$syzygy(f_1, \ldots, f_k): \ \boldsymbol{b} = (s_1, \ldots, s_k) \in R^k, \ \ with \ \sum_{i=1,\ldots,k} s_i f_i \ = \ 0$

$\boldsymbol{b_{ij}}: \quad a\, t\, g_i - b\, s\, g_j \ = \ Spol(g_i, g_j) \ = \ \sum_{i=1,\ldots k} s_i g_i \ = \ 0$

$B_G \ = \ \{ \ \boldsymbol{b_{ij}} = (s_{1\mathrm{ij}}, \ldots, s_{kij}) \mid Spol(g_i, g_j) \ \rightarrow_G \ 0 \ \}, \quad B_F \ = \ \begin{pmatrix} I - \hat{G}\,\hat{F} \\ B_G\,\hat{G} \end{pmatrix}$

# Applications (3)

- ideal quotient
- radical ideal

$J = id(H), H, F \subset R, H = \{h_1, \ldots, h_m\}$

$quotient \quad J:F \;=\; \{\; g \in R \mid g\,f \in J \quad for\,all\; f \in F \;\}$

$proposition \quad J:F \;=\; \cap_{f \in F}\, J:f$

$\boldsymbol{S} \;=\; syzygy(f, h_1, \ldots, h_m), \;\; Q \;=\; \{\; q \mid (q, q_1, \ldots, q_m) \in \boldsymbol{S} \;\}$

$proposition \quad id(H):f \;=\; id(Q)$

$radical \quad \sqrt{J} \;=\; \{\; g \in R \mid g^s \in J \quad for\,some\; s \in \mathbb{N} \;\}$

$f \in \sqrt{id(F)} \;\Leftrightarrow\; id(F \cup \{1 - t\,f\}) \cap K[X] \;\neq\; \emptyset, \;\; X \ll t$

# Applications (4)

- residue classes
- intersection of residue classes, interpolation
- dimension of ideals

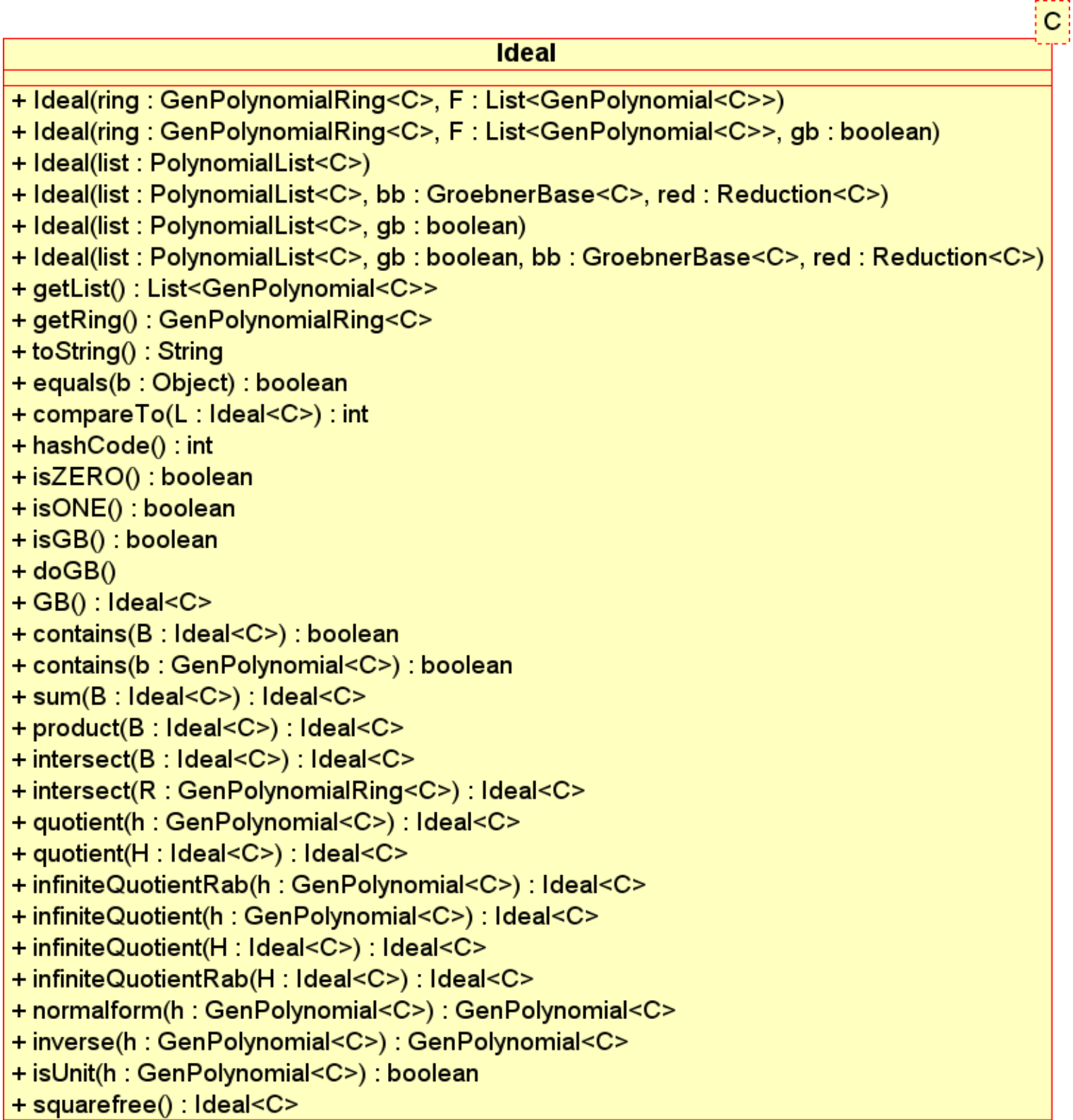
$G, F \subset R, G = GB(F)$

$A_G = \{ \bar{h} \mid f \to_G \bar{h}, h \ \ irreducible, f \in R \}$

$proposition \ \ A_G \ \ represents\, residue\, classes\, mod \ \ id(F)$

$h \in A_G \ \ is\, invertible\, mod \ \ id(F) \Leftrightarrow$
$\text{exists}\, g, g_1, \ldots, g_k \in R \ \ with \ \ h g + \sum_{i=1,\ldots,k} g_i f_i = 1$

$\text{test if} \ \ 1 \in G = GB(F'), F' = \{h\} \cup F, \ \ \text{take}\, g \,\text{from} \ \ extGB(F')$


C
Ideal
+ Ideal(ring : GenPolynomialRing<C>, F : List<GenPolynomial<C>>)
+ Ideal(ring : GenPolynomialRing<C>, F : List<GenPolynomial<C>>, gb : boolean)
+ Ideal(list : PolynomialList<C>)
+ Ideal(list : PolynomialList<C>, bb : GroebnerBase<C>, red : Reduction<C>)
+ Ideal(list : PolynomialList<C>, gb : boolean)
+ Ideal(list : PolynomialList<C>, gb : boolean, bb : GroebnerBase<C>, red : Reduction<C>)
+ getList() : List<GenPolynomial<C>>
+ getRing() : GenPolynomialRing<C>
+ toString() : String
+ equals(b : Object) : boolean
+ compareTo(L : Ideal<C>) : int
+ hashCode() : int
+ isZERO() : boolean
+ isONE() : boolean
+ isGB() : boolean
+ doGB()
+ GB() : Ideal<C>
+ contains(B : Ideal<C>) : boolean
+ contains(b : GenPolynomial<C>) : boolean
+ sum(B : Ideal<C>) : Ideal<C>
+ product(B : Ideal<C>) : Ideal<C>
+ intersect(B : Ideal<C>) : Ideal<C>
+ intersect(R : GenPolynomialRing<C>) : Ideal<C>
+ quotient(h : GenPolynomial<C>) : Ideal<C>
+ quotient(H : Ideal<C>) : Ideal<C>
+ infiniteQuotientRab(h : GenPolynomial<C>) : Ideal<C>
+ infiniteQuotient(h : GenPolynomial<C>) : Ideal<C>
+ infiniteQuotient(H : Ideal<C>) : Ideal<C>
+ infiniteQuotientRab(H : Ideal<C>) : Ideal<C>
+ normalform(h : GenPolynomial<C>) : GenPolynomial<C>
+ inverse(h : GenPolynomial<C>) : GenPolynomial<C>
+ isUnit(h : GenPolynomial<C>) : boolean
+ squarefree() : Ideal<C>

# Residue class class

C

**Residue**

+ Residue(r : ResidueRing<C>)
+ Residue(r : ResidueRing<C>, a : GenPolynomial<C>)
+ Residue(r : ResidueRing<C>, a : GenPolynomial<C>, u : int)
+ clone() : Residue<C>
+ isZERO() : boolean
+ isONE() : boolean
+ isUnit() : boolean
+ isConstant() : boolean
+ toString() : String
+ compareTo(b : Residue<C>) : int
+ equals(b : Object) : boolean
+ hashCode() : int
+ abs() : Residue<C>
+ sum(S : Residue<C>) : Residue<C>
+ negate() : Residue<C>
+ signum() : int
+ subtract(S : Residue<C>) : Residue<C>
+ divide(S : Residue<C>) : Residue<C>
+ inverse() : Residue<C>
+ remainder(S : Residue<C>) : Residue<C>
+ multiply(S : Residue<C>) : Residue<C>
+ monic() : Residue<C>
+ gcd(b : Residue<C>) : Residue<C>
+ egcd(b : Residue<C>) : Residue<C>[]

C

**ResidueRing**

+ ResidueRing(i : Ideal<C>)
+ copy(c : Residue<C>) : Residue<C>
+ getZERO() : Residue<C>
+ getONE() : Residue<C>
+ isCommutative() : boolean
+ isAssociative() : boolean
+ isField() : boolean
+ characteristic() : java.math.BigInteger
+ fromInteger(a : java.math.BigInteger) : Residue<C>
+ fromInteger(a : long) : Residue<C>
+ toString() : String
+ equals(b : Object) : boolean
+ hashCode() : int
+ random(n : int) : Residue<C>
+ random(k : int, l : int, d : int, q : float) : Residue<C>
+ random(n : int, rnd : Random) : Residue<C>
+ parse(s : String) : Residue<C>
+ parse(r : Reader) : Residue<C>

# Example

```
List<GenPolynomial<Residue<BigRational>>> L = null;
L = new ArrayList<GenPolynomial<Residue<BigRational>>>();

BigRational bfac = new BigRational(1);
GenPolynomialRing<BigRational> pfac = null;
pfac = new GenPolynomialRing<BigRational>(bfac,1);

List<GenPolynomial<BigRational>> F = null;
F = new ArrayList<GenPolynomial<BigRational>>();

GenPolynomial<BigRational> p = null;
for ( int i = 0; i < 3; i++) {
    p = pfac.random(5,5,5,0.4f);
    F.add(p);
}
//System.out.println("F = " + F);

Ideal<BigRational> id = new Ideal<BigRational>(pfac,F);
id.doGB();
ResidueRing<BigRational> rr = new ResidueRing<BigRational>(id);
System.out.println("rr = " + rr);
```

# Example (cont.)

```
String[] vars = new String[] { "a", "b" };
GenPolynomialRing<Residue<BigRational>> fac;
fac = new GenPolynomialRing<Residue<BigRational>>(rr,2,vars);
GenPolynomial<Residue<BigRational>> pp;
for ( int i = 0; i < 2; i++) {
    pp = fac.random(2,4,6,0.2f);
    if ( !pp.isConstant() ) {
       L.add(pp);
    }
}
System.out.println("L = " + L);

GroebnerBase<Residue<BigRational>> bb
    = new GroebnerBasePseudoSeq<Residue<BigRational>>(rr);
System.out.println("isGB(L) = " + bb.isGB(L));

List<GenPolynomial<Residue<BigRational>>> G = null;
G = bb.GB(L);
System.out.println("G = " + G);
System.out.println("isGB(G) = " + bb.isGB(G));

ComputerThreads.terminate();
```

# Example (output)

```
rr = ResidueRing[ BigRational( x0, x1 ) IGRLEX
(
( x0^4 + 169/192 x0 + 13/152  ),
( x1^4 - 22/15 x1 - 1/24 x0 - 63/22  )
) ]

L = [ { x0 + 24/31 } a^4 + a^3
+ { x0^2 * x1^2 + x1^2 - 27/14 x0^2 + 75/8 x0 } ,
a^2 * b^5 + { x1 + 30/17 } b^3 + { 2 } ]

isGB(L) = false

G = [ { x0 + 24/31 } a^4 + a^3
+ { x0^2 * x1^2 + x1^2 - 27/14 x0^2 + 75/8 x0 },
{ x0^2 * x1^2 + x1^2 - 27/14 x0^2 + 75/8 x0 } b^5
+ { - x0 * x1 + 24/31 x1 + 30/17 x0 + 720/527 } a^2 * b^3
+ { - x1 + 30/17 } a * b^3 + { - 2 x0 + 48/31 } a^2 - { 2 } a ]

isGB(G) = true
```

# Conclusions

- consistent object oriented design and implementation of a library for algebraic computations
- Java advantages
  - generic types for type safety
  - multi-threaded, networked, 64-bit ready
  - dynamic memory management
- benefits for computer algebra
  - non-trivial structures can be implemented
  - library can be used from any Java program

# Thank you

- Questions?
- Comments?
- http://krum.rz.uni-mannheim.de/jas
- http://krum.rz.uni-mannheim.de/kredel/ca-sem-2008.pdf
- Thanks to
  - Thomas Becker
  - Aki Yoshida
  - Raphael Jolly
  - many others