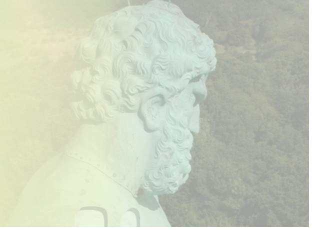
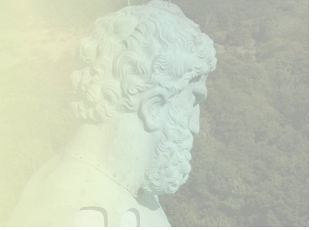


Algebraic structures as typed objects

Heinz Kredel, University of Mannheim
Raphael Jolly, Databeans

CASC 2011, Kassel



Overview

Introduction

Algebraic structures as typed objects

Ring elements and ring factories, algorithms and factories

Algebraic and transcendental extensions

Real algebraic numbers and complex algebraic numbers

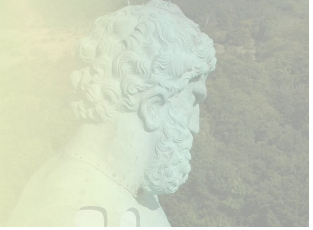
Algebraic structures in scripting interpreters

Problems

Generic types and subclasses

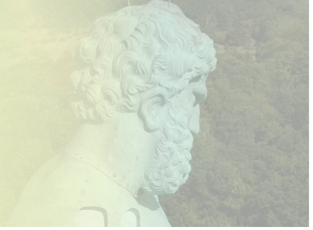
Dependent types

Conclusions



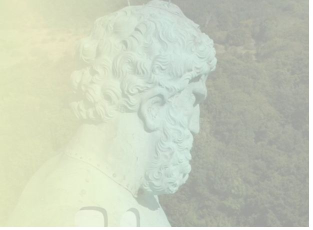
Introduction

- Software architecture for computer algebra systems :
  - run-time infrastructure, memory management parallel hardware support
  - statically typed object oriented algorithm libraries
  - dynamic interactive scripting interpreters
- reuse existing projects – concentrate on algebra, design and implementation
- be reused : Meditor, Symja, MathPiper, GeoGebra



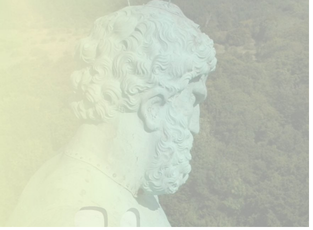
Need for types

- Scratchpad, Axiom, Aldor
- Kenzo : algebraic topology, object oriented with run-time type safety
- MuPad : object oriented layer with 'categories'
- DoCon : field extension towers, type safe, Haskell
- Pros and cons of our approach
  - see (related) work in Jolly & Kredel, CASC 2010



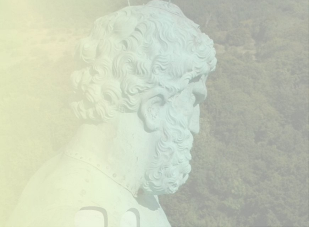
Field (and ring) extensions

- K computable field (or ring), e.g. prime fields
  - rational numbers $\mathbb{Q}$
  - modular integers $\mathbb{Z}_m, \mathbb{Z}_p$
- algebraic extensions $K(\alpha)=K[x]_{/(f)}, with\, f(\alpha)=0$
- transcendental extensions $K(x)$
- real algebraic extensions $\mathbb{Q}(+\sqrt{2})$
- complex algebraic extensions $\mathbb{Q}(+i)$



Design and implementation

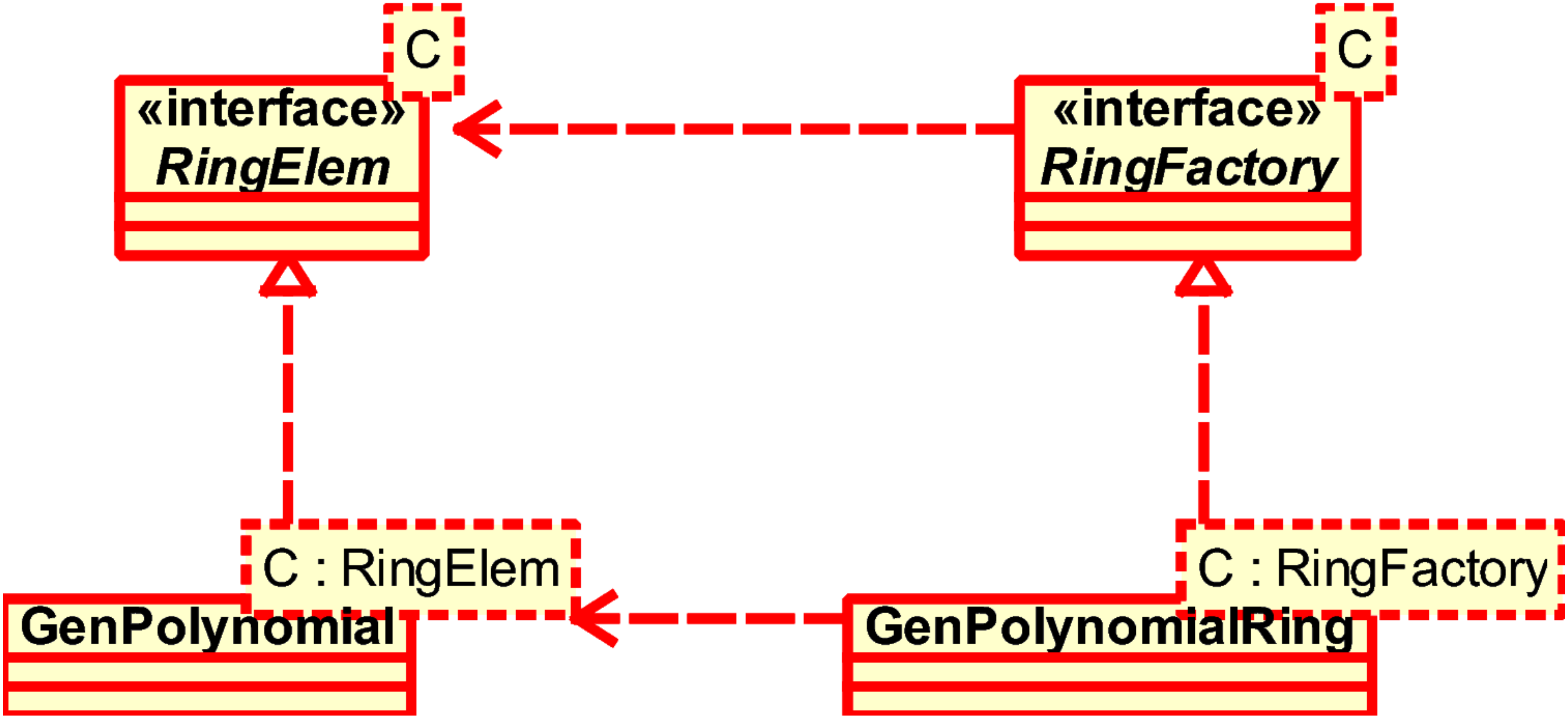
- Goal : design and implement extensions so that they can be coefficient rings of polynomial rings and relevant properties are preserved
- provide algorithms so that polynomials over real algebraic extensions have real root isolation
- provide algorithms so that polynomials over complex algebraic extensions have complex root isolation
  - fundamental theorem of algebra
  - constructive version : Weierstraß-Durand-Kerner fixpoint method



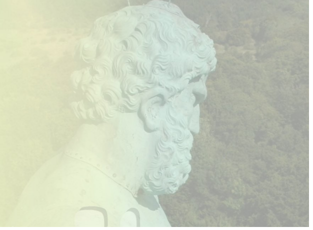
Overview

- Introduction
- **Algebraic structures as typed objects**
  - Ring elements and ring factories, algorithms and factories
  - Algebraic and transcendental extensions
  - Real algebraic numbers and complex algebraic numbers
  - Algebraic structures in scripting interpreters
- Problems
  - Generic types and subclasses
  - Dependent types
- Conclusions

Basic types



factory() method provides a back-link



Example

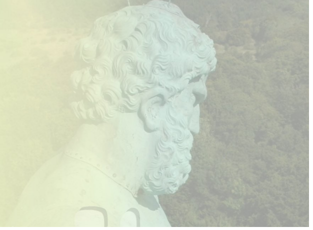
$w^2 - 2 \in \mathbb{Q}[w]$:

```
BigRational rf = new BigRational(1); // element = factory

GenPolynomialRing<BigRational> pf

= new GenPolynomialRing<BigRational>(rf,new String[]{"w"});

GenPolynomial<BigRational> a = pf.parse("w^2 - 2");
```



Algorithms and factories

example univariate Hensel lifting :

$$((\mathbb{Z}[x],a),(\mathbb{Z}_p[x],(a_1,\ldots,a_r)),(\mathbb{N},k))\rightarrow(\mathbf{Z}_{p^k}[\mathbf{x}],(b_1,\ldots,b_r))$$

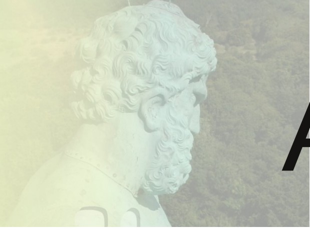
meaning :

$$(a\in\mathbb{Z}[x],(a_1,\ldots,a_r)\in\mathbb{Z}_p[x]^r,k\in\mathbb{N})\rightarrow(b_1,\ldots,b_r)\in\mathbb{Z}_{p^k}[x]^r$$

using type annotations :

$$(a:\mathbb{Z}[x],(a_1,\ldots,a_r):\mathbb{Z}_p[x]^r,k:\mathbb{N})\rightarrow(b_1,\ldots,b_r):\mathbb{Z}_{p^k}[x]^r$$

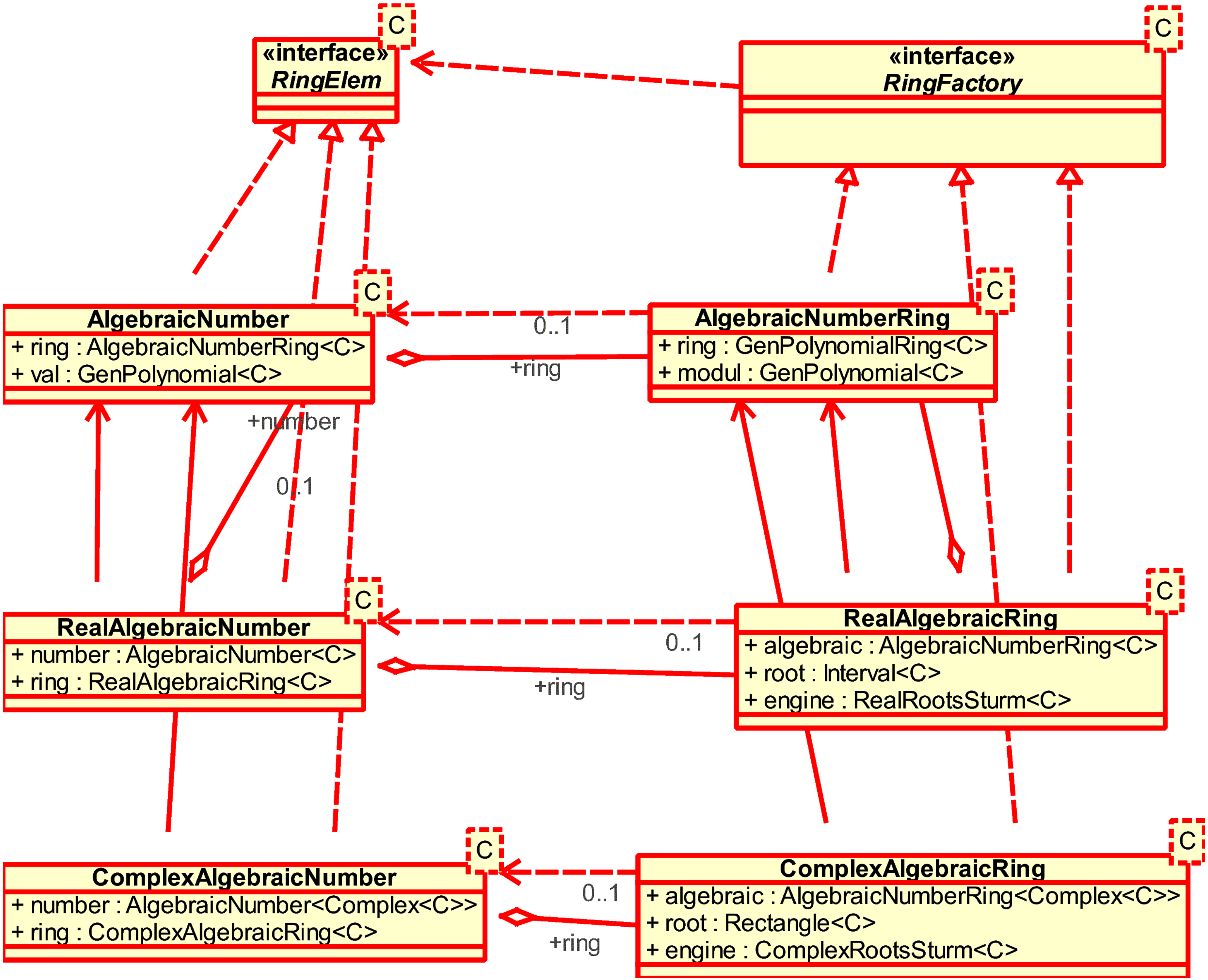
the last ring is constructed within the algorithm

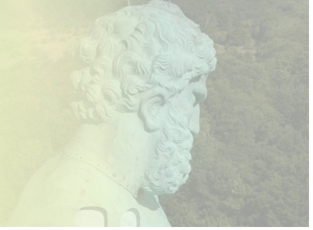


Algebraic and transcendental ring and field extensions

- $L = K(\alpha) = K[x]_{/(f)}, \mathit{with}\, f(\alpha) = 0, \mathit{is\ field\ iff}\ f\ \mathit{is\ irreducible}$
  - AlgebraicNumber, AlgebraicNumberRing
    - better names : AlgebraicElement, AlgebraicExtensionRing
- $L = K(x) = \{ \ \frac{p}{q} \ : \ p, q \ \in \ K[x], \ q > 0, \ gcd(p, q) = 1 \ \}$
  - Quotient, QuotientRing
- the construction works for all computable fields as base fields
  - so towers of field / ring extensions can be constructed
  - for example $\mathbb{Q}(\sqrt{2})(x)(\sqrt{x})$

Algebraic numbers





Example construction (1)

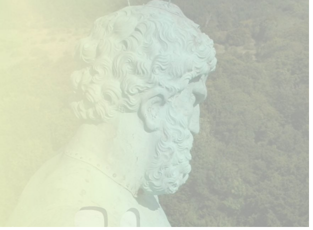
$$\mathbb{Q}(\sqrt{2})(x)(\sqrt{x})$$

$$\mathbb{Q} \rightarrow_1 \mathbb{Q}[w] \rightarrow_2 \mathbb{Q}[w]_{/(w^2-2)} \rightarrow_3 (\mathbb{Q}[w]_{/(w^2-2)})(x)$$

$$\rightarrow_4 (\mathbb{Q}[w]_{/(w^2-2)})(x)[wx] \rightarrow_5 (\mathbb{Q}[w]_{/(w^2-2)})(x)[wx]_{/(wx^2-x)}$$

```
AlgebraicNumber<Quotient<AlgebraicNumber<BigRational>>> elem;

elem = fac.parse("wx + x^5");
```



Example construction (2)

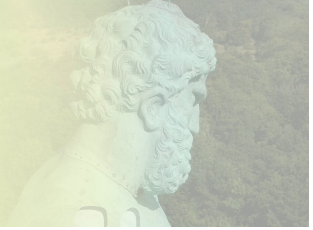
```
GenPolynomial<BigRational> a = pf.parse("w^2 - 2");
AlgebraicNumberRing<BigRational> af
  = new AlgebraicNumberRing<BigRational>(a);

String[] vx = new String[]{ "x" };
GenPolynomialRing<AlgebraicNumber<BigRational>> tf
  = new GenPolynomialRing<AlgebraicNumber<BigRational>>(af,vx);
QuotientRing<AlgebraicNumber<BigRational>> qf
  = new QuotientRing<AlgebraicNumber<BigRational>>(tf);

String[] vw = new String[]{ "wx" };
GenPolynomialRing<Quotient<AlgebraicNumber<BigRational>>> qaf
  = new
GenPolynomialRing<Quotient<AlgebraicNumber<BigRational>>>(qf,vw);

GenPolynomial<Quotient<AlgebraicNumber<BigRational>>> b
  = qaf.parse("wx^2 - x");
AlgebraicNumberRing<Quotient<AlgebraicNumber<BigRational>>> fac
  = new AlgebraicNumberRing<Quotient<AlgebraicNumber<BigRational>>>(b);
```

can be avoided with Java 7



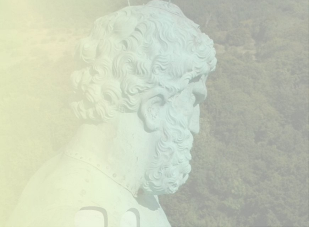
Extension field builder

- above construction is tedious but exact
- much 'boiler plate' code
- Scala can spare some type annotations via type resolution
- more simplification using 'builder pattern'
  - for example

```
RingFactory fac = ExtensionFieldBuilder
                  .baseField(new BigRational(1))
                  .algebraicExtension("w", "w^2 - 2")
                  .transcendentExtension("x")
                  .algebraicExtension("wx", "wx^2 - x")
                  .build();
```

Applications and optimizations

- such field towers can be used as coefficients for polynomial rings
- then computations like Gröbner bases can be performed in these polynomial rings
- can use primitive elements for multiple extensions
- `build()` method to optimize the extension towers
  - structural optimizations
    - transcendental high, algebraic lower in tower
    - or residue class ring modulo a Gröbner base
  - simplification
    - simple extension via primitive element (CAD example)



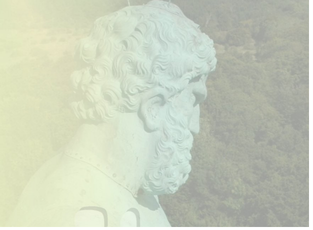
Real algebraic numbers

$$K(\alpha) \;=\; K[x]_{/(f)}, \;\; with\, f(\alpha)=0, \;\; \alpha \in \mathbb{R}, \;\; char(K)=0$$

$$I=[l,r] \subset \mathbb{R} \;\; isolating\ interval\, for\, \alpha:$$

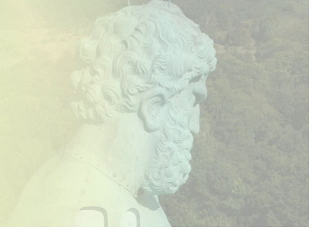
$$\alpha \in I\, for\ exactly\ one\ real\ root\ \alpha\ of\ f$$

- implementation using delegation to algebraic extension ring
  - sub-classing not possible, see 'problems' later
- classes `RealAlgebraicNumber` with factory `RealAlgebraicRing`
  - factory contains isolating interval and root engine



Real root computation

- using Sturm sequences
  - faster algorithms are future work
- classes RealRootAbstract and RealRootsSturm
- one generic implementation for any real field tower
- can construct polynomials over such fields
  - GenPolynomial<RealAlgebraicNumber<BigRational>>
- can continue with real roots for such polynomials
  - RealRootsSturm<RealAlgebraicNumber<BigRational>>
  - method realSign() used in signum() method
  - unique feature to our knowledge



Example

$$L=\mathbb{Q}(+\sqrt[3]{3})(+\sqrt{+\sqrt[3]{3}})(+\sqrt[5]{2}),\quad I=[1,2]$$

```
fac = ExtensionFieldBuilder
        .baseField(new BigRational())
        .realAlgebraicExtension("q", "q^3 - 3","[1,2]")
        .realAlgebraicExtension("w", "w^2 - q","[1,2]")
        .realAlgebraicExtension("s", "s^5 - 2","[1,2]")
        .build();
```

$$y^2-\sqrt{+\sqrt[3]{3}}\cdot\sqrt[5]{2}\ \in\ L[y]$$

Decimal approximation of the two real roots with 50 decimal digits

```
-1.1745272686769866126436905900619307101229226521299
 1.1745272686769866126436905900619307101229226521299
```

1.2 sec, approximation to 50 digits 5.2 sec, AMD at 3 GHz, IcedTea6 JVM

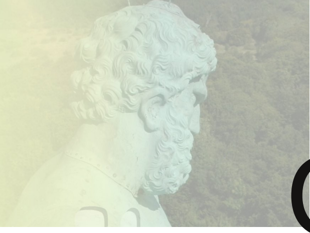
Complex algebraic numbers (1)

$$K(\gamma) \;=\; K[x]_{/(f)}, \;\; with\, f(\gamma)=0, \;\; \gamma \in \mathbb{C}, \;\; char(K)=0$$

$$I=[l_r, r_r]\times[l_i, r_i] \subset \mathbb{C} \;\; isolating\ rectangle\, for\, \gamma:$$

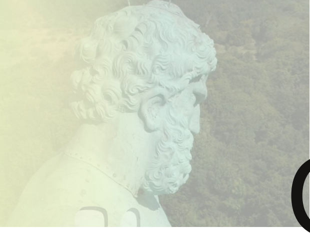
$$\gamma \in I\, for\ exactly\ one\ complex\ root\ \gamma\ of\ f$$

- classes `ComplexAlgebraicNumber` with factory `ComplexAlgebraicRing` in `edu.jas.root`
  - factory contains isolating rectangle
  - roots work only for a single extension, no towers
  - since real or imaginary parts cannot be extracted
  - need bi-variate representation



Complex root computation (1)

- using Sturm sequences, and a method derived from Wilf's numeric Routh-Hurwitz method
  - faster algorithms are future work
- classes `ComplexRootsAbstract` `ComplexRootsSturm`
- can construct polynomials over such fields
  - `GenPolynomial<ComplexAlgebraicNumber<BigRational>>`
- cannot continue with complex roots for such polynomials
  - ~~`ComplexRootsSturm<ComplexAlgebraicNumber<.>>`~~



Complex root computation (2)

- alternative : represent as tuples of real roots of the ideal generated by the equations for the real and imaginary part
- repr. as extension of two real algebraic numbers
- one implementation for any complex field tower

$$z \rightarrow a+bi \ \text{ in } \ f(z)=f(a,b)=f_r(a,b)+f_i(a,b)i$$

$$\gamma \in L = L'(i), \ \ with f(\gamma)=0$$

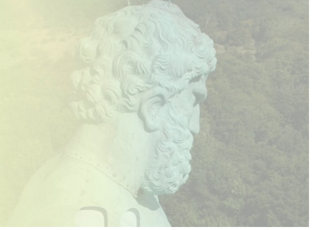
$$\gamma = \alpha + \beta i, \ \ with \ \ f_r(\alpha,\beta)=f_i(\alpha,\beta)=0$$

$$L'=K(\alpha,\beta), \ \ \alpha,\beta \in \mathbb{R}, \ \ Ideal(f_r,f_i)=\{g(x),h(x,y)\}$$

$$L'=K(\alpha)(\beta), \ \ \alpha \in \mathbb{R}, \ \ \beta_{poly} \in K(\alpha)[y]$$

Complex algebraic numbers (2)

- new classes RealAlgebraicNumber and RealAlgebraicRing in edu.jas.application
  - use as Complex<RealAlgebraicNumber<.>>
- bi-variate ideal with real root tuples as input, from ideal real roots computation
  - Ideal.zeroDimRootDecomposition()
  - PolyUtilApp. realAlgebraicRoots()
- can construct polynomials over such fields
  - GenPolynomial<Complex<RealAlgebraicNumber<.>>>
- one level instantiation is also possible
  - ComplexRootsSturm<RealAlgebraicNumber<.>>



Example

$$L=\mathbb{Q}(\gamma)=\mathbb{Q}(\sqrt[3]{2})=\mathbb{Q}(\alpha,\beta,i),\quad I=[-1,-1/2]\times[1,2]$$

$$\gamma=\alpha+\beta i \quad with \quad \alpha^3+1/4=0 \quad \text{and} \quad \beta^2-3\alpha^2=0$$

$$\gamma\approx-0.6299605+1.0911236\, i$$

$$f(t)=f(t,\gamma)=t^3-\gamma^2 \;\in\; \mathbb{Q}(\gamma)[t],\quad f(\tau)=0,$$

$$f(t,\alpha,\beta,i)=t^3+2\alpha^2-2\alpha\beta i$$

$$\tau_1\approx\quad 0.8936130-0.7498304\, i,$$

$$\tau_2\approx-1.0961787-0.3989764\, i,$$

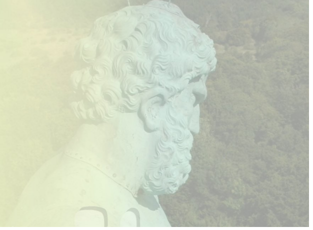
$$\tau_3\approx\quad 0.2025656+1.1488068\, i$$

$$\tau_3=\zeta+\eta i=\zeta+(256/27\,\alpha\beta\zeta^7-112/27\,\beta\zeta^4+356/27\,\alpha^2\beta\zeta)i$$

$$-10368\,\alpha\beta\zeta^9+3888\,\beta\zeta^6-15552\,\alpha^2\beta\zeta^3-81\,\alpha\beta \;=\; 0$$

$$27\,\beta\eta+192\,\zeta^7+336\,\alpha^2\zeta^4+267\,\alpha\zeta \;=\; 0$$

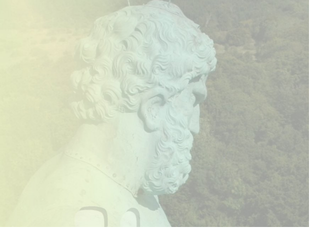
163 sec, AMD at 3 GHz, IcedTea6 JVM



Root factory

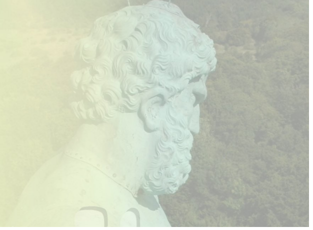
RootFactory
+ realAlgebraicNumbers(f : GenPolynomial<C>) : List<RealAlgebraicNumber<C>> + realAlgebraicNumbersField(f : GenPolynomial<C>) : List<RealAlgebraicNumber<C>> + complexAlgebraicNumbers(f : GenPolynomial<C>) : List<ComplexAlgebraicNumber<C>> + complexAlgebraicNumberComplex(f : GenPolynomial<Complex<C>>) : List<ComplexAlgebraicNumber<C>>

- link between the computation and the structures
- roots of polynomials represented as
  - list of real algebraic numbers
  - list of complex algebraic numbers
  - rings accessible via `factory()` method
- versions for fields (irreducible generator) or non fields (squarefree generator only)
- version for polynomial with complex coefficients



Algebraic structures in scripting interpreters

- Use general purpose scripting language as DSL for computer algebra
- Algebraic expressions are written in the host language ≠ strings ¿
- No need to parse (in Java)
- Type-safe (partly at run-time)
- for details see Jolly & Kredel 2008, 2009



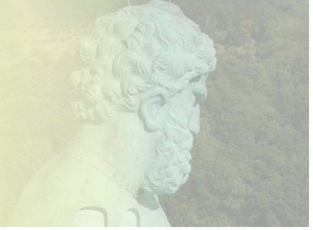
Jython

```
Q = PolyRing(QQ(),"w2",PolyRing.lex); [e,w2] = Q.gens();
Q2 = AN(w2**2 – 2,field=True);
Qp = PolyRing(Q2,"x",PolyRing.lex);
Qr = RF(Qp);
Qwx = PolyRing(Qr,"wx",PolyRing.lex);
[ewx,wwx,ax,wx] = Qwx.gens();
Q2x = AN(wx**2 – ax,field=True);
Yr = PolyRing(Q2x,"y",PolyRing.lex)
[e,w2,x,wx,y] = Yr.gens();

         EF(QQ()).extend("w2","w2^2 – 2")
             .extend("x").extend("wx","wx^2 - x").build().

f = ( y**2 - x ) * ( y**2 - 2 );
// = y**4 - ( x + 2 ) * y**2 + 2 * x
factor(f) :
// ( y - wx ) * ( y - w2 ) * ( y + wx ) * ( y + w2 )
```

9.5 seconds, 5.7 seconds after JIT warm-up, AMD 3 GHz, IcedTea6 JVM

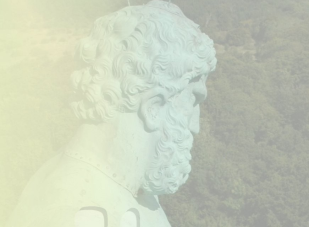


Jython : target design

- Problem : each definition of ring/extension field factory must redefine all generators in the factory tower
- Need a mechanism to « lift » values to the correct level in the ring/field tower
- Not yet fully implemented

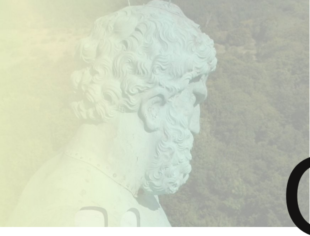
```
p = PolyRing(QQ, ["w2"]) ; [w2] = p.gens()
q = RF(PolyRing(AN(w2**2 – 2), ["x"])) ; [x] = q.gens()
r = PolyRing(q, ["wx"]) ; [wx] = r.gens()
s = PolyRing(AN(wx**2 – x), ["y"]) ; [y] = s.gens()

factor(( y**2 - x ) * ( y**2 - 2 ))
# ( y - wx ) * ( y - w2 ) * ( y + wx ) * ( y + w2 )
```



Overview

- Introduction
- Algebraic structures as typed objects
- Ring elements and ring factories, algorithms and factories
- Algebraic and transcendental extensions
- Real algebraic numbers and complex algebraic numbers
- Algebraic structures in scripting interpreters
- **Problems**
- Generic types and subclasses
- Dependent types
- Conclusions



Generic types and subclasses

- Why subclassing ?
  - Allows to reuse code of algorithms
- Problem :

```
class AlgebraicNumber<C extends GcdRingElem<C>>
implements GcdRingElem<AlgebraicNumber<C>>

class RealAlgebraicNumber<C extends GcdRingElem<C>>
extends AlgebraicNumber<C> implements
GcdRingElem<RealAlgebraicNumber<C>>
```

not possible because of type-erasure

Generic types and subclasses : delegation

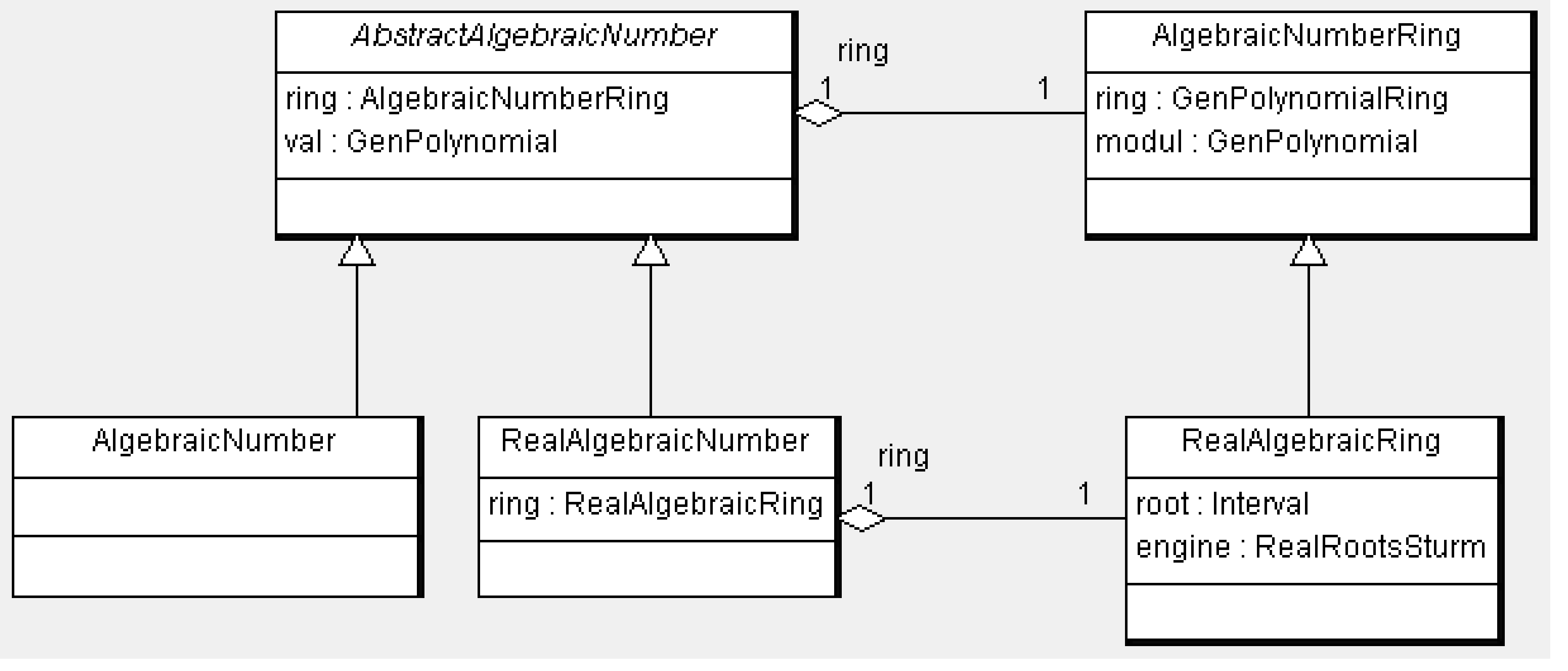
- Delegation : object features are not inherited but available through associated object (delegate)

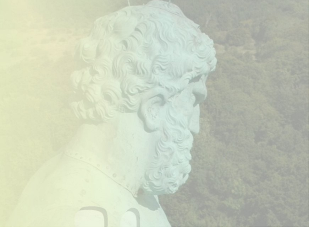
```
class RealAlgebraicNumber<C extends GcdRingElem<C>>
implements GcdRingElem<RealAlgebraicNumber<C>> {
   public final AlgebraicNumber<C> number;
}
```

- Avoids the above type-erasure problem
- Can not use algorithms written specifically for `AlgebraicNumbers` with `RealAlgebraicNumbers`

Third solution

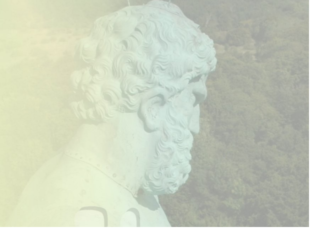
- use neither delegation nor subclassing
- inherit from a common, abstract superclass





Dependent types

- Goal : forbid operations between kinds different only with respect to some parameter
  - Integer : Mod(**7**)
  - Array of String : Polynomial(BigInt, **["x"]**)
  - Polynomial : AlgebraicNumber(**w**2 - 2**)
- Need for a dependent type
- Scala has such a concept
- Work in progress



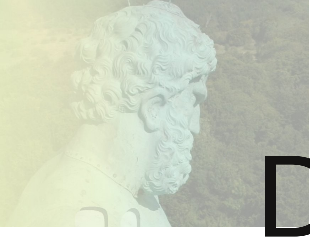
Dependent types in Scala

```
val r = Mod(7)
r(4)+r(4) // 1
val s = Mod(2)
r(4)+s(1) // problem : this works
```

→ with "val", r and s are of the same type (class)

```
object r extends Mod(7)
r(4)+r(4) // 1
object s extends Mod(2)
r(4)+s(1) // type mismatch, as expected
```

→ with "object", each value has its own type (singleton)

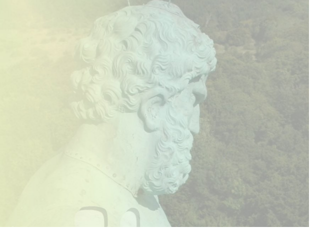


Dependent types : polynomials

```
class Polynomial[C <: Ring, P](val ring: C,
                               val variables: Array[String],
                               val ordering: Comparator[P]) {
  type E // the type of the elements of the ring
  ...
}
```

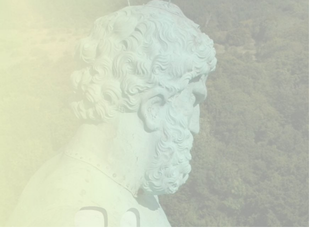
- Can use implicit conversion to lift values to correct level

```
implicit object r extends Mod(7)
implicit object p extends Polynomial(r, Array("x")) ; val
Array(x) = p.generators
implicit object q extends Polynomial(p, Array("y")) ; val
Array(y) = q.generators
// and so on
```



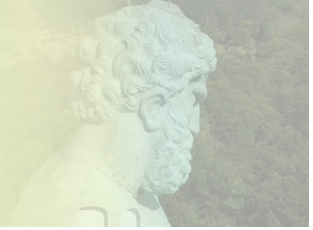
Conclusions (1)

- extensions are designed and implemented so that they can be coefficient rings of polynomial rings and relevant properties are preserved
- obtain pluggable algebraic objects by well defined interface for ring elements
- precise and explicit construction of extensions
- one generic implementation for a real root computation algorithm for any real extension field tower
- one generic implementation for a complex root computation algorithm



Conclusions (2)

- scripting languages can be used to write (run-time) type-safe algebraic expressions
- tedious work can be reduced with Scala
- dependend types can be designed in Scala
- engineering and usage of algorithm libraries benefits from type safety
- provides a Java CAS library under GPL or LGPL
- future work
  - study Scala possibilities
  - implement some faster algorithms



Thank you for your attention

Questions ?

Comments ?

http://jscl-meditor.sourceforge.net/

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

Acknowledgments

- thanks to: Thomas Becker, Werner K. Seiler, Axel Kramer, Dongming Wang, Thomas Sturm, Hans-Günther Kruse, Markus Aleksy
- thanks to the referees