dirichlet.spad

Martin Rubey

July 14, 2013

Abstract

The domain defined in this file models the Dirichlet ring,

Contents

1 The Dirichlet Ring

The Dirichlet Ring is the ring of arithmetical functions

$$f : \mathbb{N}_+ \to R$$

(see `http://en.wikipedia.org/wiki/Arithmetic_function`) together with the Dirichlet convolution (see `http://en.wikipedia.org/wiki/Dirichlet_convolution`) as multiplication and component-wise addition. Since we can consider the values an arithmetic functions assumes as the coefficients of a Dirichlet generating series, we call R the coefficient ring of a function.

In general we only assume that the coefficient ring R is a ring. If R happens to be commutative, then so is the Dirichlet ring, and in this case it is even an algebra.

Apart from the operations inherited from those categories, we only provide some convenient coercion functions.

2 domain DIRRING DirichletRing

⟨*domain DIRRING DirichletRing*⟩≡

```
)abbrev domain DIRRING DirichletRing
++ Author: Martin Rubey
++ Description: DirichletRing is the ring of arithmetical functions
```

```
++ with Dirichlet convolution as multiplication
DirichletRing(Coef: Ring):
            Exports == Implementation where

    PI ==> PositiveInteger
    FUN ==> PI -> Coef

    Exports ==> Join(Ring, Eltable(PI, Coef)) with

        if Coef has CommutativeRing then
            IntegralDomain

        if Coef has CommutativeRing then
            Algebra Coef

        coerce: FUN -> %
        coerce: % -> FUN
        coerce: Stream Coef -> %
        coerce: % -> Stream Coef

        zeta: constant -> %
        ++ zeta() returns the function which is constantly one

        multiplicative?: % -> Boolean
        ++ multiplicative?(a) returns true if the first
        ++ $streamCount$Lisp coefficients of a are multiplicative

        additive?: % -> Boolean
        ++ additive?(a) returns true if the first
        ++ $streamCount$Lisp coefficients of a are additive

    Implementation ==> add

        Rep := Record(function: FUN)

        per(f: Rep): % == f pretend %
        rep(a: %): Rep == a pretend Rep

        elt(a: %, n: PI): Coef ==
            f: FUN := (rep a).function
            f n

        coerce(a: %): FUN == (rep a).function

        coerce(f: FUN): % == per [f]
```

```
indices: Stream Integer
        := integers(1)$StreamTaylorSeriesOperations(Integer)

coerce(a: %): Stream Coef ==
    f: FUN := (rep a).function
    map((n: Integer): Coef +-> f(n::PI), indices)
       $StreamFunctions2(Integer, Coef)

coerce(f: Stream Coef): % ==
    ((n: PI): Coef +-> f.(n::Integer))::%

coerce(f: %): OutputForm == f::Stream Coef::OutputForm

1: % ==
    ((n: PI): Coef +-> (if one? n then 1$Coef else 0$Coef))::%

0: % ==
    ((n: PI): Coef +-> 0$Coef)::%

zeta: % ==
    ((n: PI): Coef +-> 1$Coef)::%

(f: %) + (g: %) ==
    ((n: PI): Coef +-> f(n)+g(n))::%

- (f: %) ==
    ((n: PI): Coef +-> -f(n))::%

(a: Integer) * (f: %) ==
    ((n: PI): Coef +-> a*f(n))::%

(a: Coef) * (f: %) ==
    ((n: PI): Coef +-> a*f(n))::%

import IntegerNumberTheoryFunctions

(f: %) * (g: %) ==
    conv := (n: PI): Coef +-> _
         reduce((a: Coef, b: Coef): Coef +-> a + b, _
                [f(d::PI) * g((n quo d)::PI) for d in divisors(n::Integer)], 0
                $ListFunctions2(Coef, Coef)
    conv::%

unit?(a: %): Boolean == not (recip(a(1$PI))$Coef case "failed")

qrecip: (%, Coef, PI) -> Coef
```

```
qrecip(f: %, f1inv: Coef, n: PI): Coef ==
    if one? n then f1inv
    else
        -f1inv * reduce(_+, [f(d::PI) * qrecip(f, f1inv, (n quo d)::PI) _
                             for d in rest divisors(n)], 0) _
                        $ListFunctions2(Coef, Coef)

recip f ==
    if (f1inv := recip(f(1$PI))$Coef) case "failed" then "failed"
    else
        mp := (n: PI): Coef +-> qrecip(f, f1inv, n)

        mp::%::Union(%, "failed")

multiplicative? a ==
    n: Integer := _$streamCount$Lisp
    for i in 2..n repeat
        fl := factors(factor i)$Factored(Integer)
        rl := [a.(((f.factor)::PI)^((f.exponent)::PI)) for f in fl]
        if a.(i::PI) ~= reduce((r:Coef, s:Coef):Coef +-> r*s, rl)
        then
            output(i::OutputForm)$OutputPackage
            output(rl::OutputForm)$OutputPackage
            return false
    true

additive? a ==
    n: Integer := _$streamCount$Lisp
    for i in 2..n repeat
        fl := factors(factor i)$Factored(Integer)
        rl := [a.(((f.factor)::PI)^((f.exponent)::PI)) for f in fl]
        if a.(i::PI) ~= reduce((r:Coef, s:Coef):Coef +-> r+s, rl)
        then
            output(i::OutputForm)$OutputPackage
            output(rl::OutputForm)$OutputPackage
            return false
    true
```

3 License

⟨*license*⟩≡

```
--Copyright (c) 2010, Martin Rubey <Martin.Rubey@math.uni-hannover.de>
--
--Redistribution and use in source and binary forms, with or without
--modification, are permitted provided that the following conditions are
--met:
--
--    - Redistributions of source code must retain the above copyright
--      notice, this list of conditions and the following disclaimer.
--
--    - Redistributions in binary form must reproduce the above copyright
--      notice, this list of conditions and the following disclaimer in
--      the documentation and/or other materials provided with the
--      distribution.
--
--THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
--IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
--TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
--PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
--OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
--EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
--PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
--PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
--LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
--NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
--SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

⟨*⟩≡

⟨*license*⟩

⟨*domain DIRRING DirichletRing*⟩