$SPAD/src/algebra derham.spad

Larry A. Lambe

12/02/90

Abstract

LALG LeftAlgebra The category of all left algebras over an arbitrary ring

EAB ExtAlgBasis This domain represents the set of all ordered subsets of the set X

ANTISYM AntiSymm The domain of antisymmetric polynomials

DERHAM DeRhamComplex The deRham complex of Euclidean space

Contents

1 category LALG LeftAlgebra

⟨*category LALG LeftAlgebra*⟩≡

```
)abbrev category LALG LeftAlgebra
++ Author: Larry A. Lambe
++ Date  : 03/01/89; revised 03/17/89; revised 12/02/90.
++ Description: The category of all left algebras over an arbitrary
++ ring.
LeftAlgebra(R:Ring): Category == Join(Ring, LeftModule R) with
    --operations
      coerce: R -> %
        ++ coerce(r) returns r * 1 where 1 is the identity of the
        ++ left algebra.
    add
      coerce(x:R):% == x * 1$%
```

2 domain EAB ExtAlgBasis

⟨*domain EAB ExtAlgBasis*⟩≡

```
)abbrev domain EAB ExtAlgBasis
--% ExtAlgBasis
++  Author: Larry Lambe
++  Date created: 03/14/89
++  Description:
++  A domain used in the construction of the exterior algebra on a set
++  X over a ring R.  This domain represents the set of all ordered
++  subsets of the set X, assumed to be in correspondance with
++  {1,2,3, ...}.  The ordered subsets are themselves ordered
++  lexicographically and are in bijective correspondance with an ordered
++  basis of the exterior algebra.  In this domain we are dealing strictly
++  with the exponents of basis elements which can only be 0 or 1.
--  Thus we really have L({0,1}).
++
++  The multiplicative identity element of the exterior algebra corresponds
++  to the empty subset of X.  A coerce from List Integer to an
++  ordered basis element is provided to allow the convenient input of
++  expressions. Another exported function forgets the ordered structure
++  and simply returns the list corresponding to an ordered subset.

ExtAlgBasis(): Export == Implement where
   I   ==> Integer
   L   ==> List
   NNI ==> NonNegativeInteger

   Export == OrderedSet with
     coerce     : L I -> %
        ++ coerce(l) converts a list of 0's and 1's into a basis
        ++ element, where 1 (respectively 0) designates that the
        ++ variable of the corresponding index of l is (respectively, is not)
        ++ present.
        ++ Error: if an element of l is not 0 or 1.
     degree     : %   -> NNI
        ++ degree(x) gives the numbers of 1's in x, i.e., the number
        ++ of non-zero exponents in the basis element that x represents.
     exponents  : %   -> L I
        ++ exponents(x) converts a domain element into a list of zeros
        ++ and ones corresponding to the exponents in the basis element
        ++ that x represents.
--   subscripts : %   -> L I
        -- subscripts(x) looks at the exponents in x and converts
        -- them to the proper subscripts
     Nul        : NNI -> %
```

```
        ++ Nul() gives the basis element 1 for the algebra generated
        ++ by n generators.

   Implement == add
     Rep := L I
     x,y :  %

     x = y == x =$Rep y

     x < y ==
       null x            => not null y
       null y            => false
       first x = first y => rest x < rest y
       first x > first y

     coerce(li:(L I)) ==
       for x in li repeat
         if x ~= 1 and x ~= 0 then error "coerce: values can only be 0 and 1"
       li

     degree x         == (_+/x)::NNI

     exponents x      == copy(x @ Rep)

--   subscripts x     ==
--      cntr:I := 1
--      result: L I := []
--      for j in x repeat
--        if j = 1 then result := cons(cntr,result)
--        cntr:=cntr+1
--      reverse_! result

     Nul n            == [0 for i in 1..n]

     coerce x         == coerce(x @ Rep)$(L I)
```

3 domain ANTISYM AntiSymm

⟨*domain ANTISYM AntiSymm*⟩≡

```
)abbrev domain ANTISYM AntiSymm
++   Author: Larry A. Lambe
++   Date     : 01/26/91.
++   Revised  : 30 Nov 94
++
++   based on AntiSymmetric '89
++
++   Needs: ExtAlgBasis, FreeModule(Ring,OrderedSet), LALG, LALG-
++
++   Description: The domain of antisymmetric polynomials.


AntiSymm(R:Ring, lVar:List Symbol): Export == Implement where
  LALG ==> LeftAlgebra
  FMR  ==> FM(R,EAB)
  FM   ==> FreeModule
  I    ==> Integer
  L    ==> List
  EAB  ==> ExtAlgBasis     -- these are exponents of basis elements in order
  NNI  ==> NonNegativeInteger
  O    ==> OutputForm
  base ==> k
  coef ==> c
  Term ==> Record(k:EAB,c:R)

  Export == Join(LALG(R), RetractableTo(R)) with
      leadingCoefficient : %           -> R
        ++ leadingCoefficient(p) returns the leading
        ++ coefficient of antisymmetric polynomial p.
--    leadingSupport       : %           -> EAB
      leadingBasisTerm     : %           -> %
        ++ leadingBasisTerm(p) returns the leading
        ++ basis term of antisymmetric polynomial p.
      reductum           : %           -> %
        ++ reductum(p), where p is an antisymmetric polynomial,
        ++ returns p minus the leading
        ++ term of p if p has at least two terms, and 0 otherwise.
      coefficient        : (%,%)     -> R
        ++ coefficient(p,u) returns the coefficient of
        ++ the term in p containing the basis term u if such
        ++ a term exists, and 0 otherwise.
        ++ Error: if the second argument u is not a basis element.
      generator          : NNI         -> %
```

```
        ++ generator(n) returns the nth multiplicative generator,
        ++ a basis term.
      exp               : L I         -> %
        ++  exp([i1,...in]) returns \spad{u_1\^{i_1} ... u_n\^{i_n}}
      homogeneous?      : %           -> Boolean
        ++  homogeneous?(p) tests if all of the terms of
        ++  p have the same degree.
      retractable?      : %           -> Boolean
        ++  retractable?(p) tests if p is a 0-form,
        ++  i.e., if degree(p) = 0.
      degree            : %           -> NNI
        ++  degree(p) returns the homogeneous degree of p.
      map               : (R -> R, %) -> %
        ++  map(f,p) changes each coefficient of p by the
        ++  application of f.


--    1 corresponds to the empty monomial Nul = [0,...,0]
--    from EAB.  In terms of the exterior algebra on X,
--    it corresponds to the identity element which lives
--    in homogeneous degree 0.

  Implement == FMR add
      Rep := L Term
      x,y :  EAB
      a,b :  %
      r   :  R
      m   :  I

      dim := #lVar

      1 == [[ Nul(dim)$EAB, 1$R ]]

      coefficient(a,u) ==
        not null u.rest => error "2nd argument must be a basis element"
        x := u.first.base
        for t in a repeat
          if t.base = x then return t.coef
          if t.base < x then return 0
        0

      retractable?(a) ==
        null a or (a.first.k  =  Nul(dim))

      retractIfCan(a):Union(R,"failed") ==
        null a               => 0$R
```

```
    a.first.k = Nul(dim) => leadingCoefficient a
    "failed"

  retract(a):R ==
    null a => 0$R
    leadingCoefficient a

  homogeneous? a ==
    null a => true
    siz := _+/exponents(a.first.base)
    for ta in reductum a repeat
      _+/exponents(ta.base) ~= siz => return false
    true

  degree a ==
    null a => 0$NNI
    homogeneous? a => (_+/exponents(a.first.base)) :: NNI
    error "not a homogeneous element"

  zo : (I,I) -> L I
  zo(p,q) ==
    p = 0 => [1,q]
    q = 0 => [1,1]
    [0,0]

  getsgn : (EAB,EAB) -> I
  getsgn(x,y) ==
    sgn:I  := 0
    xx:L I := exponents x
    yy:L I := exponents y
    for i in 1 .. (dim-1) repeat
      xx  := rest xx
      sgn := sgn + (_+/xx)*yy.i
    sgn rem 2 = 0 => 1
    -1

  Nalpha: (EAB,EAB) -> L I
  Nalpha(x,y) ==
    i:I := 1
    dum2:L I := [0 for i in 1..dim]
    for j in 1..dim repeat
      dum:=zo((exponents x).j,(exponents y).j)
      (i := i*dum.1) = 0 => break
      dum2.j := dum.2
    i = 0 => cons(i, dum2)
    cons(getsgn(x,y), dum2)
```

```
      a * b ==
        null a => 0
        null b => 0
        ((null a.rest) and (a.first.k = Nul(dim))) => a.first.c * b
        ((null b.rest) and (b.first.k = Nul(dim))) => b.first.c * a
        z:% := 0
        for tb in b repeat
          for ta in a repeat
            stuff:=Nalpha(ta.base,tb.base)
            r:=first(stuff)*ta.coef*tb.coef
            if r ~= 0 then z := z + [[rest(stuff)::EAB, r]]
        z

      coerce(r):% ==
        r = 0 => 0
        [ [Nul(dim), r] ]

      coerce(m):% ==
        m = 0 => 0
        [ [Nul(dim), m::R] ]

      characteristic() == characteristic()$R

      generator(j) ==
        -- j < 1 or j > dim => error "your subscript is out of range"
        -- error will be generated by dum.j if out of range
        dum:L I := [0 for i in 1..dim]
        dum.j:=1
        [[dum::EAB, 1::R]]

      exp(li:(L I)) ==  [[li::EAB, 1]]

      leadingBasisTerm a ==
        [[a.first.k, 1]]

      displayList:EAB -> O
      displayList(x):O ==
        le: L I := exponents(x)$EAB
--      reduce(_*,[(lVar.i)::O for i in 1..dim | le.i = 1])$L(O)
--        reduce(_*,[(lVar.i)::O for i in 1..dim | one?(le.i)])$L(O)
        reduce(_*,[(lVar.i)::O for i in 1..dim | ((le.i) = 1)])$L(O)

      makeTerm:(R,EAB) -> O
      makeTerm(r,x) ==
      -- we know that r ~= 0
```

```
        x = Nul(dim)$EAB  => r::O
--        one? r => displayList(x)
        (r = 1) => displayList(x)
--      r = 1 => displayList(x)
--      r = 0 => 0$I::O
--      x = Nul(dim)$EAB  => r::O
        r::O * displayList(x)

      coerce(a):O ==
        zero? a     => 0$I::O
        null rest(a @ Rep) =>
                 t := first(a @ Rep)
                 makeTerm(t.coef,t.base)
        reduce(_+,[makeTerm(t.coef,t.base) for t in (a @ Rep)])$L(O)
```

4 domain DERHAM DeRhamComplex

⟨*domain DERHAM DeRhamComplex*⟩≡

```
)abbrev domain DERHAM DeRhamComplex
++ Author: Larry A. Lambe
++ Date    : 01/26/91.
++ Revised : 12/01/91.
++
++ based on code from '89 (AntiSymmetric)
++
++ Needs: LeftAlgebra, ExtAlgBasis, FreeMod(Ring,OrderedSet)
++
++ Description: The deRham complex of Euclidean space, that is, the
++ class of differential forms of arbitary degree over a coefficient ring.
++ See Flanders, Harley, Differential Forms, With Applications to the Physical
++ Sciences, New York, Academic Press, 1963.

DeRhamComplex(CoefRing,listIndVar:List Symbol): Export == Implement where
  CoefRing :  Join(Ring, Comparable)
  ASY     ==> AntiSymm(R,listIndVar)
  DIFRING ==> DifferentialRing
  LALG    ==> LeftAlgebra
  FMR     ==> FreeMod(R,EAB)
  I       ==> Integer
  L       ==> List
  EAB     ==> ExtAlgBasis  -- these are exponents of basis elements in order
  NNI     ==> NonNegativeInteger
  O       ==> OutputForm
  R       ==> Expression(CoefRing)

  Export == Join(LALG(R), RetractableTo(R)) with
      leadingCoefficient : %           -> R
        ++ leadingCoefficient(df) returns the leading
        ++ coefficient of differential form df.
      leadingBasisTerm   : %           -> %
        ++ leadingBasisTerm(df) returns the leading
        ++ basis term of differential form df.
      reductum           : %           -> %
        ++ reductum(df), where df is a differential form,
        ++ returns df minus the leading
        ++ term of df if df has two or more terms, and
        ++ 0 otherwise.
      coefficient        : (%,%)     -> R
        ++ coefficient(df,u), where df is a differential form,
        ++ returns the coefficient of df containing the basis term u
        ++ if such a term exists, and 0 otherwise.
```

```
    generator          : NNI          -> %
      ++ generator(n) returns the nth basis term for a differential form.
    homogeneous?       : %            -> Boolean
      ++  homogeneous?(df) tests if all of the terms of
      ++  differential form df have the same degree.
    retractable?       : %            -> Boolean
      ++  retractable?(df) tests if differential form df is a 0-form,
      ++  i.e., if degree(df) = 0.
    degree             : %            -> I
      ++  degree(df) returns the homogeneous degree of differential form df.
    map                : (R -> R, %) -> %
      ++  map(f,df) replaces each coefficient x of differential
      ++  form df by \spad{f(x)}.
    totalDifferential    : R -> %
      ++  totalDifferential(x) returns the total differential
      ++  (gradient) form for element x.
    exteriorDifferential : % -> %
      ++  exteriorDifferential(df) returns the exterior
      ++  derivative (gradient, curl, divergence, ...) of
      ++  the differential form df.

  Implement == ASY add
    Rep := ASY

    dim := #listIndVar

    totalDifferential(f) ==
      divs:=[differentiate(f,listIndVar.i)*generator(i)$ASY for i in 1..dim]
      reduce("+",divs)

    termDiff : (R, %) -> %
    termDiff(r,e) ==
      totalDifferential(r) * e

    exteriorDifferential(x) ==
      x = 0 => 0
      termDiff(leadingCoefficient(x)$Rep,leadingBasisTerm x) + exteriorDifferential(

    lv := [concat("d",string(liv))$String::Symbol for liv in listIndVar]

    displayList:EAB -> O
    displayList(x):O ==
      le: L I := exponents(x)$EAB
--    reduce(_*,[(lv.i)::O for i in 1..dim | le.i = 1])$L(O)
--      reduce(_*,[(lv.i)::O for i in 1..dim | one?(le.i)])$L(O)
      reduce(_*,[(lv.i)::O for i in 1..dim | ((le.i) = 1)])$L(O)
```

```
      makeTerm:(R,EAB) -> O
      makeTerm(r,x) ==
      -- we know that r ~= 0
        x = Nul(dim)$EAB  => r::O
--        one? r => displayList(x)
        (r = 1) => displayList(x)
--      r = 1 => displayList(x)
        r::O * displayList(x)

      terms : % -> List Record(k: EAB, c: R)
      terms(a) ==
        -- it is the case that there are at least two terms in a
        a pretend List Record(k: EAB, c: R)

      coerce(a):O ==
        a           = 0$Rep => 0$I::O
        ta := terms a
--      reductum(a) = 0$Rep => makeTerm(leadingCoefficient a, a.first.k)
        null ta.rest => makeTerm(ta.first.c, ta.first.k)
        reduce(_+,[makeTerm(t.c,t.k) for t in ta])$L(O)
```

5 License

⟨*license*⟩≡

⟨***⟩≡

⟨*license*⟩

⟨*category LALG LeftAlgebra*⟩
⟨*domain EAB ExtAlgBasis*⟩
⟨*domain ANTISYM AntiSymm*⟩
⟨*domain DERHAM DeRhamComplex*⟩

References

[1] nothing