\$SPAD/src/algebra product.spad

The Axiom Team

July 17, 2011

Abstract

This domain implements cartesian product for a pair of (possibly different) domains. If the underlying domains are both Finite then the resulting Product is also Finite and can be enumerated via size(), index(), location(), etc. The index of the second component (B) varies most quickly.

Contents

1 domain PRODUCT Product

⟨*domain PRODUCT Product*⟩≡

```
)abbrev domain PRODUCT Product
++ Description:
++ This domain implements cartesian product
Product (A:SetCategory,B:SetCategory) : C == T
 where
  C == SetCategory  with
       if A has Finite and B has Finite then Finite
       if A has Monoid and B has Monoid then Monoid
       if A has AbelianMonoid and B has AbelianMonoid then AbelianMonoid
       if A has CancellationAbelianMonoid and
          B has CancellationAbelianMonoid then CancellationAbelianMonoid
       if A has Group  and B has Group  then  Group
       if A has AbelianGroup and B has AbelianGroup then  AbelianGroup
       if A has OrderedAbelianMonoidSup and B has OrderedAbelianMonoidSup
                                           then OrderedAbelianMonoidSup
       if A has OrderedSet and B has OrderedSet then  OrderedSet

       makeprod     : (A,B) -> %
        ++ makeprod(a,b) \undocumented
       selectfirst  :   %   -> A
        ++ selectfirst(x) \undocumented
       selectsecond :   %   -> B
        ++ selectsecond(x) \undocumented

  T == add

    --representations
       Rep := Record(acomp:A,bcomp:B)

    --declarations
       x,y: %
       i: NonNegativeInteger
       p: NonNegativeInteger
       a: A
       b: B
       d: Integer

    --define
       coerce(x):OutputForm == paren [(x.acomp)::OutputForm,
                                      (x.bcomp)::OutputForm]
       x=y ==
           x.acomp = y.acomp => x.bcomp = y.bcomp
           false
```

```
makeprod(a:A,b:B) :%   == [a,b]

selectfirst(x:%) : A   == x.acomp

selectsecond (x:%) : B == x.bcomp

if A has Monoid and B has Monoid then
   1 == [1$A,1$B]
   x * y == [x.acomp * y.acomp,x.bcomp * y.bcomp]
   x ** p == [x.acomp ** p ,x.bcomp ** p]

if A has Finite and B has Finite then
   size == size$A * size$B
   index(n) == [index((((n::Integer-1) quo size$B )+1)::PositiveInteger)$A,
                index((((n::Integer-1) rem size$B )+1)::PositiveInteger)$B]
   random() == [random()$A,random()$B]
   lookup(x) == ((lookup(x.acomp)$A::Integer-1) * size$B::Integer + lookup(x.bcomp)$B::Integer)::PositiveInteger
   hash(x) == hash(x.acomp)$A * size$B::SingleInteger + hash(x.bcomp)$B

if A has Group and B has Group then
   inv(x) == [inv(x.acomp),inv(x.bcomp)]

if A has AbelianMonoid and B has AbelianMonoid then
   0 == [0$A,0$B]

   x + y == [x.acomp + y.acomp,x.bcomp + y.bcomp]

   c:NonNegativeInteger * x == [c * x.acomp,c*x.bcomp]

if A has CancellationAbelianMonoid and
   B has CancellationAbelianMonoid then
     subtractIfCan(x, y) : Union(%,"failed") ==
       (na:= subtractIfCan(x.acomp, y.acomp)) case "failed" => "failed"
       (nb:= subtractIfCan(x.bcomp, y.bcomp)) case "failed" => "failed"
       [na::A,nb::B]

if A has AbelianGroup and B has AbelianGroup then
   - x == [- x.acomp,-x.bcomp]
   (x - y):% == [x.acomp - y.acomp,x.bcomp - y.bcomp]
   d * x == [d * x.acomp,d * x.bcomp]

if A has OrderedAbelianMonoidSup and B has OrderedAbelianMonoidSup then
   sup(x,y) == [sup(x.acomp,y.acomp),sup(x.bcomp,y.bcomp)]

if A has OrderedSet and B has OrderedSet then
   x < y ==
```

```
                xa:= x.acomp ; ya:= y.acomp
                xa < ya => true
                xb:= x.bcomp ; yb:= y.bcomp
                xa = ya => (xb < yb)
                false

--     coerce(x:%):Symbol ==
--      PrintableForm()
--      formList([x.acomp::Expression,x.bcomp::Expression])$PrintableForm
```

2 License

⟨*license*⟩≡


```
--Copyright (c) 1991-2002, The Numerical ALgorithms Group Ltd.
--All rights reserved.
--
--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.
--
--    - Neither the name of The Numerical ALgorithms Group Ltd. nor the
--      names of its contributors may be used to endorse or promote products
--      derived from this software without specific prior written permission.
--
--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.
```

$\langle * \rangle \equiv$
$\langle$*license*$\rangle$

$\langle$*domain PRODUCT Product*$\rangle$

References

[1] nothing