

tilings

Claude Heiland-Allen

2011–2015

Contents

1	Data/Tiling/Class.hs	2
2	Data/Tiling.hs	4
3	Data/Tiling/Quad.hs	5
4	.gitignore	6
5	LICENSE	6
6	Setup.hs	7
7	tilings.cabal	7

1 Data/Tiling/Class.hs

```
{-# LANGUAGE DeriveDataTypeable #-}
{- |
Module      : Data.Tiling.Class
Copyright   : (c) Claude Heiland-Allen 2011
5  License   : BSD3

Maintainer  : claud@mathr.co.uk
Stability   : unstable
Portability : portable
10

Substitution tiling API.
-}
module Data.Tiling.Class where

15 import Data.Data (Data)
import Data.Typeable (Typeable)
import Data.List (partition)

-- | Substitution tilings. Instances must obey the following laws:
20 --
--     > parent root == Nothing
--     > all (== Just t) . map parent . children $ t
--     > t 'inside' exterior t
--     > t 'encloses' interior t
25 --     > interior t 'insideR' exterior t
--     > t 'inside' r ==> t 'overlaps' r
--     > t 'encloses' r ==> t 'overlaps' r
--     > t 'overlaps' r ==> not (t 'outside' r)
--     > t 'encloses' r && n >= 0 ==> not $ any ('outside' r) (tile t r n)
30 --
-- Minimal complete definition: all except 'tile'.
class Tiling t where
    -- | The largest tile to start from.
```

```

root      :: t
35  -- | The smaller children of a tile.
children  :: t -> [t]
-- | The unique parent of a tile.
parent    :: t -> Maybe t
-- | A rectangle that completely encloses the tile.
40  exterior :: t -> Rectangle
-- | A rectangle that is completely enclosed by the tile.
interior  :: t -> Rectangle
-- | Test if a rectangle completely encloses the tile.
inside    :: t -> Rectangle -> Bool
45  -- | Test if a rectangle is completely enclosed by the tile.
encloses  :: t -> Rectangle -> Bool
-- | Test if a rectangle is completely disjoint from the tile.
outside   :: t -> Rectangle -> Bool
-- | Test if a rectangle has any overlap with the tile.
50  overlaps :: t -> Rectangle -> Bool
-- | Generate a tiling that completely fills the given rectangle.
--
--   Preconditions:
--
55  --   > t 'encloses' r
--   > n >= 0
--
tile      :: t -> Rectangle -> Int -> [t]
tile      = tileDefault
60
-- | Default implementation for 'tile'.
tileDefault :: Tiling t => t -> Rectangle -> Int -> [t]
tileDefault t r n
  | n >= 0 = uncurry (++) $ iterate step ([t], []) !! n
65  | otherwise = error "Data.Tiling.Class.tileDefault: not (n >= 0)"
  where
    step (es, is) =
      let is' = concatMap children is
          es' = concatMap children es
70          (is'', es'') = partition ('inside' r) . filter ('overlaps' r) $ es'
          in (es'', is'' ++ is')

-- | An axis-aligned rectangle with 'Rational' coordinates.
--
75  --   Invariant:
--
--   > westEdge r <= eastEdge r && southEdge r <= northEdge r
--
--   For substitution tilings that contain irrational lengths and/or scale
80  --   factors, the intention is that the implementations of 'exterior',
--   and 'interior' provide reasonably tight bounds, within a percent
--   or two, say, while the data type maintains full precision internally
--   (perhaps using algebraic field extensions over 'Rational').
data Rectangle = Rectangle{ northEdge, southEdge, eastEdge, westEdge :: !ℤ
  ↳ Rational }
85  deriving (Eq, Ord, Read, Show, Data, Typeable)

-- | Create a valid rectangle, sorting the edges to meet the invariant.
rectangle :: Rational {- ^ x0 -} -> Rational {- ^ x1 -} -> Rational {- ^ y0 -} ↳
  ↳ -> Rational {- ^ y1 -} -> Rectangle {- ^ rectangle -}

```

```

rectangle x0 x1 y0 y1 = Rectangle
90   { northEdge = y0 'max' y1, southEdge = y0 'min' y1
    , eastEdge  = x0 'max' x1, westEdge  = x0 'min' x1
    }

-- | Check if a rectangle is inside another rectangle. The comparison
95 -- is not strict, so that a rectangle is inside itself.
insideR :: Rectangle -> Rectangle -> Bool
insideR p q
    = northEdge p <= northEdge q
    && southEdge p >= southEdge q
100   && eastEdge  p <= eastEdge  q
    && westEdge  p >= westEdge  q

-- | Check if a rectangle is disjoint from another rectangle. The comparison
-- is strict, so that neighbouring rectangles that share an edge will
105 -- not be outside each other.
outsideR :: Rectangle -> Rectangle -> Bool
outsideR p q
    = northEdge p < southEdge q
    || southEdge p > northEdge q
110   || eastEdge  p < westEdge  q
    || westEdge  p > eastEdge  q

-- | Check if a rectangle overlaps with another rectangle. The comparison
-- is not strict, so that neighbouring rectangles that share an edge
115 -- will overlap each other.
overlapsR :: Rectangle -> Rectangle -> Bool
overlapsR p q = not (p 'outsideR' q)

```

2 Data/Tiling.hs

```

{- |
Module      : Data.Tiling
Copyright   : (c) Claude Heiland-Allen 2011
License     : BSD3
5
Maintainer  : claud@mathr.co.uk
Stability   : unstable
Portability : portable

10 Substitution tilings. The term substitution, in connection with tilings,
describes a simple but powerful method to produce tilings with many
interesting properties.

The main idea is to use a finite set of building blocks called prototiles,
15 an expanding linear map (the inflation factor), and a rule how to dissect
each scaled tile into copies of the original prototiles.

For some examples of substitution tilings, and a glossary of terminology,
see the /tilings encyclopedia/
20 at <http://tilings.math.uni-bielefeld.de/>
-}
module Data.Tiling
( module Data.Tiling.Class
  , module Data.Tiling.Quad
25 ) where

```

```
import Data.Tiling.Class
import Data.Tiling.Quad
```

3 Data/Tiling/Quad.hs

```
{-# LANGUAGE DeriveDataTypeable #-}
{- |
Module      : Data.Tiling.Quad
Copyright    : (c) Claude Heiland-Allen 2011
5  License   : BSD3

Maintainer   : claud@mathr.co.uk
Stability    : unstable
Portability   : portable
10

Simple substitution tiling with each square divided into four quadrants
(with no rotation).
-}
module Data.Tiling.Quad
15  ( Quadrant(..), isNorth, isSouth, isWest, isEast, quadrants
    , Quad(..), quadChild, quadParent, quadPath, quadFile
    , module Data.Tiling.Class
    ) where

20  import Data.Data (Data)
import Data.Typeable (Typeable)
import Data.Bits (bit, shiftL, shiftR, testBit, (.|.))
import Data.List (unfoldr)
import Data.Ratio ((%))
25

import Data.Tiling.Class

-- | A square tile.
data Quad = Quad{ quadLevel :: !Int, quadWest, quadNorth :: !Integer }
30  deriving (Read, Show, Eq, Ord, Data, Typeable)

-- | Substitution tiling for square tiles.
instance Tiling Quad where
instance Tiling Quad where
    root = Quad 0 0 0
35    children q = map ('quadChild' q) quadrants
    parent q = snd 'fmap' quadParent q
    exterior (Quad l x y) =
        let d = bit l
        in  rectangle (x % d) ((x + 1) % d) (y % d) ((y + 1) % d)
40    interior = exterior
    inside q r = exterior q 'insideR' r
    encloses q r = r 'insideR' interior q
    outside q r = exterior q 'outsideR' r
    overlaps q r = exterior q 'overlapsR' r
45

-- | Which quadrant.
data Quadrant = NorthWest | NorthEast | SouthWest | SouthEast
    deriving (Read, Show, Eq, Ord, Enum, Bounded, Data, Typeable)

50  isNorth, isSouth, isWest, isEast :: Quadrant -> Bool
isEast c = fromEnum c 'testBit' 0
```

```

isSouth c = fromEnum c `testBit` 1
isNorth = not . isSouth
isWest = not . isEast
55
-- | All quadrants.
quadrants :: [Quadrant]
quadrants = [minBound .. maxBound]

60
-- | The child tile at a given quadrant.
quadChild :: Quadrant -> Quad -> Quad
quadChild c Quad{ quadLevel = l, quadWest = x, quadNorth = y } = Quad
  { quadLevel = l + 1
    , quadWest = x `shiftL` 1 .|. (fromIntegral . fromEnum . isEast) c
65    , quadNorth = y `shiftL` 1 .|. (fromIntegral . fromEnum . isSouth) c
  }

-- | The parent with quadrant information for the tile. Satisfies:
--
70
-- > quadParent (quadChild c q) == Just (c, q)
quadParent :: Quad -> Maybe (Quadrant, Quad)
quadParent Quad{ quadLevel = l, quadWest = x, quadNorth = y }
  | l > 0 = Just
    ( toEnum (fromEnum (y `testBit` 0) `shiftL` 1 .|. fromEnum (x `testBit` 0))
      ↪ )
75    , Quad{ quadLevel = l - 1, quadWest = x `shiftR` 1, quadNorth = y `shiftR` 1
      ↪ 1 }
    )
  | otherwise = Nothing

-- | The path from this tile to the root. Satisfies:
80
-- > foldr quadChild root (quadPath q) == q
quadPath :: Quad -> [Quadrant]
quadPath = unfoldr quadParent

85
-- | Suggested file system location for data pertaining to a 'Quad'.
quadFile :: Quad -> Maybe ([FilePath], FilePath)
quadFile q
  | null cs = Nothing
  | otherwise = Just (init cs, last cs)
90
where
  -- based on a suggestion from Robert Munafo <http://mrob.com>.
  cs = chunk 2 . map unsafeName . chunk 2 . reverse . quadPath $ q
  unsafeName :: [Quadrant] -> Char
  unsafeName [c] = ['a'..'d'] !! (fromEnum c)
  unsafeName [c,d] = ['e'..'t'] !! (fromEnum c `shiftL` 2 .|. fromEnum d)
95  unsafeName _ = error "Data.Tiling.Quad.quadFile.unsafeName"
  chunk :: Int -> [a] -> [[a]]
  chunk _ [] = []
  chunk n xs = let (ys, zs) = splitAt n xs in ys : chunk n zs

```

4 .gitignore

dist

5 LICENSE

Copyright (c)2011, Claude Heiland-Allen

All rights reserved.

5 Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

10 * 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.

15 * Neither the name of Claude Heiland-Allen nor the names of other
contributors may be used to endorse or promote products derived
from this software without specific prior written permission.

20 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
25 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
30 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6 Setup.hs

```
import Distribution.Simple
main = defaultMain
```

7 tilings.cabal

```
Name:           tilings
Version:        0.1
Synopsis:       substitution tilings
Description:
```

5 Substitution tilings. The term substitution, in connection with tilings,
describes a simple but powerful method to produce tilings with many
interesting properties.

 .
The main idea is to use a finite set of building blocks called prototiles,
10 an expanding linear map (the inflation factor), and a rule how to dissect
each scaled tile into copies of the original prototiles.

 .
For some examples of substitution tilings, and a glossary of terminology,
see the /tilings encyclopedia/
15 at <<http://tilings.math.uni-bielefeld.de/>>

```
Homepage:       http://code.mathr.co.uk/tilings
License:        BSD3
License-file:   LICENSE
```

```
20  Author:          Claude Heiland-Allen
    Maintainer:      claud@mathr.co.uk

    Category:        Math
    Build-type:      Simple
25  Cabal-version:    >=1.2

    Library
      Exposed-modules: Data.Tiling, Data.Tiling.Class, Data.Tiling.Quad
      Build-depends:   base >= 4 && < 6
30  GHC-options:      -Wall -fno-warn-duplicate-exports
```