hp2pretty

Claude Heiland-Allen

2010–2018

Contents

1 BUGS

```
empty heap profile gives NaN in SVG - detect and abort?
```

2 examples/Describe2.hp.gz

(application/gzip; charset=binary)

3 examples/figure.sh

```
#!/bin/bash
cat $1 |
while read svg rest
do
  re='^(.+) [(](.+)[)]$'
  if [[ $rest =~ $re ]]
  then
    title="${BASH_REMATCH[1]}"
    date="${BASH_REMATCH[2]}"
    printf '\\begin{figure}[!ht]\n'
    printf '\\centering\n'
    printf '\\caption{\\lstinline[columns=fixed]{%s} (%s)}\n' "${title}" "${date↙
        ↪ }"
    printf '\\includesvg[width=\\textwidth,pretex=\\small]{%s}\n' "${svg}"
    printf '\\end{figure}\n'
  fi
done
printf '\\begin{table}[!ht]\n'
printf '\\centering\n'
printf '\\caption{Legend}\n'
printf '\\begin{tabular}{cl}\n'
cat $2 | sed 's|\\|\\\\|g' |
while read svg label
do
  printf '\includesvg[width=0.02109375\\textwidth]{%s} & |%s| \\\\\n' "${svg}" "↙
      ↪ ${label}"
done
printf '\\end{tabular}\n'
printf '\\end{table}\n'
```

4 examples/html.sh

```
#!/bin/bash
cat <<END
<!DOCTYPE html>
<html>
<head>
<title>hp2pretty</title>
</head>
<body style="width:1280px;margin:auto;">
<h1>hp2pretty</h1>
END
cat $1 |
while read svg rest
do
  re='^(.+) [(](.+)[)]$'
  if [[ $rest =~ $re ]]
  then
    title="${BASH_REMATCH[1]}"
    date="${BASH_REMATCH[2]}"
    printf '<h2><code>%s</code> (%s)</h2>\n' "${title}" "${date}"
    printf '<img src="%s" />\n' "${svg}"
  fi
done
printf '<h2>Legend</h2>\n' "${title}" "${date}"
printf '<table>\n'
cat $2 | sed 's|\\|\\\\|g' |
```

```
while read svg label
do
  printf '<tr><td><img src="%s" /></td><td><code>%s</code></td></tr>\n' "${svg}"↙
      ↳ "${label}"
done
printf '</table>\n'
cat <<END
</body>
</html>
END
```

5 examples/Makefile

```
all: report.pdf report.html

clean:
	-rm -f Describe2.hp Describe2.svg title.txt key.txt figure.tex

Describe2.hp: Describe2.hp.gz
	gunzip < Describe2.hp.gz > Describe2.hp

Describe2.svg title.txt key.txt: Describe2.hp
	hp2pretty --title=title.txt --key=key.txt --pattern --bands=5 Describe2.↙
	    ↳ hp

figure.tex: figure.sh title.txt key.txt Describe2.svg
	bash figure.sh title.txt key.txt > figure.tex

report.pdf: report.tex figure.tex $(wildcard *.svg)
	pdflatex -synctex=1 -interaction=nonstopmode --shell-escape report.tex

report.html: html.sh title.txt key.txt Describe2.svg $(wildcard *.svg)
	bash html.sh title.txt key.txt > report.html
```

6 examples/report.tex

```
\documentclass{article}
\usepackage[margin=2cm]{geometry}
\usepackage[utf8]{inputenc}
\usepackage{lmodern}
\usepackage{svg}

\usepackage{listings}
\lstMakeShortInline[columns=fixed]|

\renewcommand{\familydefault}{\sfdefault}

\title{Report}
\author{hp2pretty}
\date{\today}

\begin{document}

\maketitle

\input{figure.tex}
```

```
\end{document}
```

7 .gitignore

```
.cabal-sandbox
cabal.sandbox.config
dist
test
*.hi
*.o
src/Main
*.svg
*.hp
*.html
*.pdf
tests/escapes
tests/namelen
tests/unicode
```

8 hp2pretty.cabal

```
Name:                hp2pretty
Version:             0.9
Synopsis:            generate pretty graphs from heap profiles
Description:         hp2pretty is a rewrite of hp2ps, implemented in Haskell, ↙
    ↳ with
                     the aims of being maintainable, with more flexible output, ↙
                         ↳ and
                     more beautiful output.  Currently hp2pretty outputs ↙
                         ↳ Scalable
                     Vector Graphics (SVG) only, though PostScript (PS) is ↙
                         ↳ planned.
                     Not all of hp2ps' options are implemented yet in hp2pretty.
                     .
                     In hp2pretty-0.9 a mode for detached key is added:
                     .
                     > hp2pretty --key=inline *.hp
                     > hp2pretty --key=key.txt *.hp
                     > hp2pretty --key=- *.hp
                     .
                     The output file is an simple text file, that mentions ↙
                         ↳ additional
                     SVG files for the legend - how you format it is up to you.
                     .
                     A mode for detached titles is also added:
                     .
                     > hp2pretty --title=inline *.hp
                     > hp2pretty --title=title.txt *.hp
                     > hp2pretty --title=- *.hp
                     .
                     The output file is an simple text file, that mentions the ↙
                         ↳ image
                     SVG files.  You could use this for figure captions, etc.
                     .
                     See the examples/ directory in the source distribution for ↙
                         ↳ hints.
```

```
                    .
                    In hp2pretty-0.8 output filtering and sorting flags are ↙
                        ↳ added,
                    as well as low-ink pattern fills for printing:
                    .
                    > hp2pretty --trace=1     *.hp
                    > hp2pretty --bands=15    *.hp
                    > hp2pretty --sort=size   *.hp
                    > hp2pretty --sort=stddev *.hp
                    > hp2pretty --sort=name   *.hp
                    > hp2pretty --reverse     *.hp
                    > hp2pretty --pattern     *.hp
                    .
                    In hp2pretty-0.7 a parsing bug is fixed.
                    .
                    In hp2pretty-0.6 ByteString is replaced by Text, fixing ↙
                        ↳ bugs
                    with Unicode.
                    .
                    In hp2pretty-0.5 using attoparsec and floatshow internally
                    should give a healthy speedup.
                    .
                    In hp2pretty-0.4 usage changed since the previous release:
                    .
                    > hp2pretty *.hp
                    > hp2pretty --uniform-scale=time   *.hp
                    > hp2pretty --uniform-scale=memory *.hp
                    > hp2pretty --uniform-scale=both   *.hp
                    .
                    Colours also changed: now they are based on a hash of the
                    cost label, which should make colours have stable semantics
                    across program runs.

Homepage:           https://mathr.co.uk/blog/hp2pretty.html
License:            BSD3
License-file:       LICENSE
Author:             Claude Heiland-Allen
Maintainer:         claude@mathr.co.uk
Copyright:          (C) 2010,2011,2015,2017,2018  Claude Heiland-Allen
Category:           Development
Build-type:         Simple
Extra-source-files:
  BUGS
  NEWS
  README
  THANKS
  examples/Describe2.hp.gz
  examples/figure.sh
  examples/html.sh
  examples/Makefile
  examples/report.tex

Cabal-version:      >=1.6

Executable hp2pretty
  Build-depends:    base >= 4 && < 5,
                    array,
```

```
                      attoparsec,
                      containers >= 0.5,
                      filepath,
                      floatshow,
                      mtl,
                      optparse-applicative,
                      semigroups,
                      text
    GHC-options:      -Wall -rtsopts
    HS-source-dirs:   src
    Main-is:          Main.hs
    Other-modules:    Args
                      Types
                      Total
                      Prune
                      Bands
                      Pretty
                      Print
                      SVG
                      Graphics
                      Pattern

Source-repository head
  type:               git
  location:           https://code.mathr.co.uk/hp2pretty.git

Source-repository this
  type:               git
  location:           https://code.mathr.co.uk/hp2pretty.git
  tag:                v0.9
```

9 LICENSE

10 NEWS

```
v0.9    2018-08-01      detach

New flags and features:
  --key=inline|FILE.txt     (optionally generate detached key)
  --title=inline|FILE.txt   (optionally generate detached title)


v0.8.0.2        2017-06-29      imports

Add imports to fix build with even older GHC versions.


v0.8.0.1        2017-06-29      semigroups

Depend on semigroups to fix build with older GHC versions.


v0.8    2017-06-29      patterns

New flags and features:
  --sort=size|stddev|name   (order bands by this property)
  --reverse                 (reverse sort order)
  --trace=1.0               (trace elements threshold percentage)
  --bands=15                (limit number of bands)
  --pattern                 (pattern fill, better for printing)


v0.7    2017-06-27      bugfix

A bugfix in header parsing (thanks to Pepe Iborra). An
adjustment to text rendering (no longer stroked, but just
filled). HTTPS for urls.


v0.6    2015-04-08      text

ByteString is replaced by Text, fixing bugs with Unicode.


v0.5    2011-10-15      acceleration

Vastly improved runtime performance thanks to 'floatshow'
and 'attoparsec' for (respectively) printing and parsing
floating point numbers.

Source code statistics: 544 lines, 3275 words, 20061 chars.
```

```
v0.4    2011-05-18      comparability

Colours are stable across program runs (based on a hash of
the label names).

Command line flag to use the same scale for all input files
(--uniform-scale=none|time|memory|both).

Usage change: specify XX.hp file(s) on the command line, the
output is to corresponding XX.svg files.

Source code statistics: 543 lines, 3273 words, 20045 chars.


v0.3    2010-11-02      special characters

Fixes a bug where broken SVG was generated when label names
contained XML-special characters.

Source code statistics: 484 lines, 2888 words, 17261 chars.


v0.2    2010-07-30      leaner and meaner

Memory usage and speed have both been improved, in part by
reading the entire file into a strict ByteString and making
two parsing passes over it, where the first pass accumulates
statistics and the second pass extracts the relevant data to
arrays, thus avoid large intermediate data structures.

Code abtraction has been improved, by separating the SVG
code into its own module and providing a Graphics interface,
so that later a PostScript implementation can be added more
easily.

Source code statistics: 475 lines, 2854 words, 17007 chars.


v0.1    2010-07-28      first release
The result of a few days hacking, hp2pretty is born.
Source code statistics: 397 lines, 2406 words, 14212 chars.
```

11 README

```
hp2pretty
=========

hp2ps is a tool to generate PostScript graphs of Heap Profiles.
hp2pretty is a rewrite of hp2ps, implemented in Haskell, with
the aims of being maintainable, with more flexible output, and
more beautiful output.  Currently hp2pretty outputs Scalable
Vector Graphics (SVG) only, though PostScript (PS) is planned.
Not all of hp2ps' options are implemented yet in hp2pretty.

Usage
=====
```

```
hp2pretty - generate pretty graphs from heap profiles

Usage: hp2pretty [--uniform-scale AXES] [--sort FIELD] [--key KEY]
                 [--title TITLE] [--reverse] [--trace PERCENT] [--bands ↙
                     ↳ COUNT]
                 [--pattern] FILES...
  Convert heap profile FILES.hp to pretty graphs FILES.svg

Available options:
  --uniform-scale AXES     Whether to use a uniform scale for all outputs. ↙
      ↳ One
                           of: none (default), time, memory, both.
  --sort FIELD             How to sort the bands. One of: size (default),
                           stddev, name.
  --key KEY                Whether to embed the key in the image output. One↙
      ↳  of:
                           inline (default), FILE.txt. Use - for standard ↙
                               ↳ output
                           and ./inline for a file named literally "inline".
  --title TITLE            Whether to embed the title in the image output. ↙
      ↳ One
                           of: inline (default), FILE.txt. Use - for ↙
                               ↳ standard
                           output and ./inline for a file named literally
                           "inline".
  --reverse                Reverse the order of bands.
  --trace PERCENT          Percentage of trace elements to
                           combine. (default: 1.0)
  --bands COUNT            Maximum number of bands to draw (0 for
                           unlimited). (default: 15)
  --pattern                Use patterns instead of solid colours to fill ↙
      ↳ bands.
  FILES...                 Heap profiles (FILE.hp will be converted to
                           FILE.svg).
  -h,--help                Show this help text
```

Benchmarks

hp2pretty-0.1

```
$ wc in.hp
 1484749  2969502 37406420 in.hp

$ time hp2pretty <in.hp >out.svg
real 0m38.786s 0m39.423s 0m38.674s
user 0m38.120s 0m38.800s 0m38.110s
sys  0m00.360s 0m00.250s 0m00.400s

$ time hp2ps <in.hp >out.ps
real 0m38.474s 0m38.765s 0m38.017s
user 0m38.350s 0m38.200s 0m37.860s
sys  0m00.090s 0m00.140s 0m00.150s
```

```
(hp2pretty compiled with "ghc -O2 --make")

Essentially identical runtime!

Memory usage from 'top':
hp2pretty ~275M
hp2ps     ~50M

Tests were run on 64bit GNU/Linux with ghc-6.12.3.


hp2pretty-0.2
-------------

"in.hp"      time (seconds)        memory (bytes)
(bytes)   hp2pretty     hp2ps   hp2pretty    hp2ps
    37M      33.345    36.430         64M      25M
   1.3M       1.015     0.074        2.2M     1.0M
    47k       0.058     0.005        210k      60k

Time is real time averaged over 3 runs as measured by 'bash'.
Memory is peak usage as measured by 'valgrind --tool=massif'.

hp2pretty compiled via cabal-install with 'ghc -O --make'.

Tests were run on 64bit GNU/Linux with ghc-6.12.3.
```

12 Setup.hs

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

13 src/Args.hs

```
module Args (args, Args(..), Uniform(..), Sort(..), KeyPlace(..), TitlePlace(..)
    ↪ ) where

import Options.Applicative
import Data.Semigroup ((<>))

data Uniform = None | Time | Memory | Both deriving (Eq)
data Sort = Size | StdDev | Name
data KeyPlace = KeyInline | KeyFile FilePath
data TitlePlace = TitleInline | TitleFile FilePath

data Args = Args
  { uniformity   :: Uniform
  , sorting      :: Sort
  , keyPlace     :: KeyPlace
  , titlePlace   :: TitlePlace
  , reversing    :: Bool
  , tracePercent :: Double
  , nBands       :: Int
  , patterned    :: Bool
  , files        :: [String]
  }
```

```
argParser :: Parser Args
argParser = Args
      <$> option parseUniform
          ( long "uniform-scale"
         <> help "Whether to use a uniform scale for all outputs.  One of: none ↙
             ↳ (default), time, memory, both."
         <> value None
         <> metavar "AXES" )
      <*> option parseSort
          ( long "sort"
         <> help "How to sort the bands.  One of: size (default), stddev, name."
         <> value Size
         <> metavar "FIELD" )
      <*> option parseKey
          ( long "key"
         <> help "Whether to embed the key in the image output.  One of: inline ↙
             ↳ (default), FILE.txt.  Use - for standard output and ./inline for ↙
             ↳ a file named literally \"inline\"."
         <> value KeyInline
         <> metavar "KEY" )
      <*> option parseTitle
          ( long "title"
         <> help "Whether to embed the title in the image output.  One of: ↙
             ↳ inline (default), FILE.txt.  Use - for standard output and ./↙
             ↳ inline for a file named literally \"inline\"."
         <> value TitleInline
         <> metavar "TITLE" )
      <*> switch
          ( long "reverse"
         <> help "Reverse the order of bands." )
      <*> option auto
          ( long "trace"
         <> help "Percentage of trace elements to combine."
         <> value 1
         <> showDefault
         <> metavar "PERCENT" )
      <*> option auto
          ( long "bands"
         <> help "Maximum number of bands to draw (0 for unlimited)."
         <> value 15
         <> showDefault
         <> metavar "COUNT" )
      <*> switch
          ( long "pattern"
         <> help "Use patterns instead of solid colours to fill bands." )
      <*> some (argument str
          ( help "Heap profiles (FILE.hp will be converted to FILE.svg)."
         <> metavar "FILES..." ))

parseUniform :: ReadM Uniform
parseUniform = eitherReader $ \s -> case s of
  "none" -> Right None
  "time" -> Right Time
  "memory" -> Right Memory
  "both" -> Right Both
  _ -> Left "expected one of: none, time, memory, both"
```

```
parseSort :: ReadM Sort
parseSort = eitherReader $ \s -> case s of
  "size" -> Right Size
  "stddev" -> Right StdDev
  "name" -> Right Name
  _ -> Left "expected one of: size, stddev, name"

parseKey :: ReadM KeyPlace
parseKey = eitherReader $ \s -> case s of
  "inline" -> Right KeyInline
  "" -> Left "expected one of: inline, FILE.txt"
  f -> Right (KeyFile f)

parseTitle :: ReadM TitlePlace
parseTitle = eitherReader $ \s -> case s of
  "inline" -> Right TitleInline
  "" -> Left "expected one of: inline, FILE.txt"
  f -> Right (TitleFile f)

args :: IO Args
args = execParser opts
  where
    opts = info (argParser <**> helper)
      ( fullDesc
     <> progDesc "Convert heap profile FILES.hp to pretty graphs FILES.svg"
     <> header "hp2pretty - generate pretty graphs from heap profiles" )
```

14 src/Bands.hs

```
{-# LANGUAGE BangPatterns #-}
module Bands (bands) where

import Control.Monad (forM_)
import Control.Monad.ST (runST)
import Data.Array.Base (unsafeFreezeSTUArray)
import Data.Array.ST (writeArray, readArray, newArray)
import Data.Array.Unboxed (UArray)
import Data.Map (Map, lookup, size)
import Prelude hiding (lookup, lines, words, length)
import Data.Text (Text, pack, unpack, lines, words, isPrefixOf, length)
import qualified Data.Text as T
import Data.Attoparsec.Text (parseOnly, double)

import Types

bands :: Header -> Map Text Int -> Text -> (UArray Int Double, UArray (Int, Int)↙
    ↳ Double)
bands h bs input = runST $ do
  times <- newArray (1, hCount h) 0
  vals  <- newArray ((-1,1), (size bs, hCount h)) 0
  forM_ (zip [1 ..] . chunkSamples . drop 4 . lines $ input) $ \(i, (t:ss)) -> ↙
      ↳ do
    writeArray times i (sampleTime sBEGIN_SAMPLE t)
    forM_ ss $ \s -> do
      let [k,vs] = words s
          !v = readDouble vs
```

```
        case k `lookup` bs of
          Nothing -> writeArray vals (0, i) . (+ v) =<< readArray vals (0, i)
          Just b  -> writeArray vals (b, i) v
    times' <- unsafeFreezeSTUArray times
    vals'  <- unsafeFreezeSTUArray vals
    return (times', vals')

chunkSamples :: [Text] -> [[Text]]
chunkSamples [] = []
chunkSamples (x:xs)
  | sBEGIN_SAMPLE `isPrefixOf` x =
      let (ys, zs) = break (sEND_SAMPLE `isPrefixOf`) xs
      in  case zs of
            [] -> [] -- discard incomplete sample
            (_:ws) -> (x:ys) : chunkSamples ws
  | otherwise = [] -- expected BEGIN_SAMPLE or EOF...

sampleTime :: Text -> Text -> Double
sampleTime name h =
  if name `isPrefixOf` h
  then readDouble . T.drop (length name + 1) $ h
  else error $ "Parse.sampleTime: expected " ++ unpack name ++ " but got " ++ ↙
      ↳ unpack h

readDouble :: Text -> Double
readDouble s = case parseOnly double s of
  Right x -> x
  _ -> error $ "Parse.readDouble: no parse " ++ unpack s

sBEGIN_SAMPLE, sEND_SAMPLE :: Text
sBEGIN_SAMPLE = pack "BEGIN_SAMPLE"
sEND_SAMPLE = pack "END_SAMPLE"
```

15 src/Graphics.hs

```
module Graphics where

import Data.Text (Text, foldl')
import Data.Char (ord)
import Data.Bits (shiftR)
import Data.Int (Int32, Int64)

import Pattern (patterns)

type Point = (Double, Double)
type Size = (Double, Double)
type Angle = Double

type FontSize = Double
data Anchor = Start | Middle | End

type StrokeWidth = Double
type Opacity = Double
data RGB = RGB Double Double Double

type PatternID = Text
```

```
data Graphics =
  Graphics
  { text     :: Maybe Angle -> Anchor -> FontSize -> Point -> [Text] -> [Text]
  , rect     :: Point -> Size -> [Text]
  , line     :: Point -> Point -> [Text]
  , polygon  :: [Point] -> [Text]
  , pattern  :: Text -> (PatternID, [Text])
  , visual   :: Maybe (Either PatternID RGB) -> Maybe Opacity -> Maybe RGB -> ↙
    ↳ Maybe StrokeWidth -> [Text] -> [Text]
  , document :: Size -> [Text] -> [Text] -> [Text]
  }

rescalePoint :: (Point,Point) -> (Point,Point) -> Point -> Point
rescalePoint ((inX0,inY0),(inX1,inY1)) ((outX0,outY0),(outX1,outY1)) (x,y) =
  let inW = inX1 - inX0
      inH = inY1 - inY0
      outW = outX1 - outX0
      outH = outY1 - outY0
  in  ((x - inX0) / inW * outW + outX0, (y - inY0) / inH * outH + outY0)

black, white :: RGB
black = RGB 0 0 0
white = RGB 1 1 1

colour :: Text -> RGB
colour s = hsvToRGB (c 0x12345) (c 0x6789a) (c 0xbcdef)
  where
    c m = fromIntegral (hashText m s) * 667 / fromIntegral (maxBound :: Int32)

hsvToRGB :: Double -> Double -> Double -> RGB
hsvToRGB h0 s0 v0 =
  let s = 0.5 + 0.5 * (s0 - (fromIntegral (floor s0 :: Int)))
      v = 0.5 + 0.5 * (v0 - (fromIntegral (floor v0 :: Int)))
      h = floor h0 :: Int
      i = h `mod` 6
      f = h0 - fromIntegral h
      p = v * (1 - s)
      q = v * (1 - s * f)
      t = v * (1 - s * (1 - f))
  in  case i of
        0 -> RGB v t p
        1 -> RGB q v p
        2 -> RGB p v t
        3 -> RGB p q v
        4 -> RGB t p v
        _ -> RGB v p q

patternIx :: Text -> Int
patternIx s = fromIntegral (hashText 0x54321 s * 555) `mod` length patterns

-- hashing functions copied and modified from BSD-style LICENSE'd
-- base-4.3.1.0:Data.HashTable (c) The University of Glasgow 2003

hashText :: Int32 -> Text -> Int32
hashText magic = foldl' f golden
  where f m c = fromIntegral (ord c) * magic + hashInt32 m
```

```
hashInt32 :: Int32 -> Int32
hashInt32 x = mulHi x golden + x

mulHi :: Int32 -> Int32 -> Int32
mulHi a b = fromIntegral (r `shiftR` 32)
  where r :: Int64
        r = fromIntegral a * fromIntegral b

golden :: Int32
golden = 1013904242
```

16 src/Main.hs

```
{-# LANGUAGE BangPatterns #-}
{-# LANGUAGE OverloadedStrings #-}
module Main (main) where

import Prelude hiding (print, readFile)
import Data.Text.IO (readFile, hPutStr, hPutStrLn)
import qualified Data.Text as T
import Control.Monad (forM, forM_, when)
import Data.List (foldl1', nub)
import Data.Tuple (swap)
import Data.Semigroup ((<>))
import System.Exit (exitSuccess)
import System.FilePath (replaceExtension)
import System.IO (withFile, IOMode(WriteMode), stdout)

import Args (args, Args(..), Uniform(..), Sort(..), KeyPlace(..), TitlePlace(..)
    ↪ )
import Total (total)
import Prune (prune, cmpName, cmpSize, cmpStdDev)
import Bands (bands)
import Pretty (pretty)
import Print (print, printKey)
import SVG (svg)
import Types (Header(..))

main :: IO ()
main = do
  a <- args
  let uniformTime = uniformity a `elem` [Time, Both]
      uniformMemory = uniformity a `elem` [Memory, Both]
      sorting' = case sorting a of
        Name -> cmpName
        Size -> cmpSize
        StdDev -> cmpStdDev
      reversing' = if reversing a then swap else id
      cmp = fst $ reversing' sorting'
      sepkey = case keyPlace a of
        KeyInline -> False
        KeyFile{} -> True
      noTitle = case titlePlace a of
        TitleInline -> False
        TitleFile{} -> True
  when (null (files a)) exitSuccess
  labelss <- if not (uniformTime || uniformMemory)
```

```
    then forM (files a) $ \file -> do
      input <- readFile file
      let (header, totals) = total input
          keeps = prune cmp (tracePercent a) (bound $ nBands a) totals
          (times, vals) = bands header keeps input
          ((sticks, vticks), (labels, coords)) = pretty header vals keeps
          outputs = print svg noTitle sepkey (patterned a) header sticks vticks ↙
              ↳ labels times coords
      withFile (replaceExtension file "svg") WriteMode $ \h -> mapM_ (hPutStr h)↙
          ↳  outputs
      return $ (header, reverse labels)
    else do
      inputs <- mapM readFile (files a)
      let hts0 = map total inputs
          (smima, vmima) = foldl1' (\((!smi, !sma), (!vmi, !vma)) ((!smi', !sma↙
              ↳ '), (!vmi', !vma')) -> ((smi`min`smi', sma`max`sma'), (vmi`min`↙
              ↳ vmi', vma`max`vma'))) . map (\(h, _) -> (hSampleRange h, ↙
              ↳ hValueRange h)) $ hts0
          hts1 | uniformTime = map (\(h, t) -> (h{ hSampleRange = smima }, t)) ↙
              ↳ hts0
               | otherwise = hts0
          hts | uniformMemory = map (\(h, t) -> (h{ hValueRange = vmima }, t)) ↙
              ↳ hts1
              | otherwise = hts1
      forM (zip3 (files a) inputs hts) $ \(file, input, (header, totals)) -> do
        let keeps = prune cmp (tracePercent a) (bound $ nBands a) totals
            (times, vals) = bands header keeps input
            ((sticks, vticks), (labels, coords)) = pretty header vals keeps
            outputs = print svg noTitle sepkey (patterned a) header sticks ↙
                ↳ vticks labels times coords
        withFile (replaceExtension file "svg") WriteMode $ \h -> mapM_ (hPutStr ↙
            ↳ h) outputs
        return $ (header, reverse labels)
  case keyPlace a of
    KeyFile keyFile -> (if keyFile == "-" then ($ stdout) else withFile keyFile ↙
        ↳ WriteMode) $ \txt -> do
      forM_ (nub $ concatMap snd labelss) $ \label -> do
        let (filename, content) = printKey svg (patterned a) label
        withFile filename WriteMode $ \h -> mapM_ (hPutStr h) content
        hPutStrLn txt (T.pack filename <> " " <> label)
    _ -> return ()
  case titlePlace a of
    TitleFile titleFile -> (if titleFile == "-" then ($ stdout) else withFile ↙
        ↳ titleFile WriteMode) $ \txt -> do
      forM_ (files a `zip` map fst labelss) $ \(file, header) -> do
        let filename = replaceExtension file "svg"
            title = hJob header <> " (" <> hDate header <> ")"
        hPutStrLn txt (T.pack filename <> " " <> title)
    _ -> return ()

bound :: Int -> Int
bound n
  | n <= 0 = maxBound
  | otherwise = n
```

17 src/Pattern.hs

```
module Pattern (patterns, Pattern) where

import Data.Tuple (swap)

type Point = (Double, Double)

type Pattern = [(Point, Point)]

loop :: [Point] -> [(Point, Point)]
loop ps = zip ps (tail (cycle ps))

patterns :: [Pattern]
patterns =
  -- short and long dashes in 4 directions
  [ [ ((5, 8), (11, 8)) ]
  , [ ((3, 8), (13, 8)) ]
  , [ ((8, 5), (8, 11)) ]
  , [ ((8, 3), (8, 13)) ]
  , [ ((5, 5), (11, 11)) ]
  , [ ((3, 3), (13, 13)) ]
  , [ ((5, 11), (11, 5)) ]
  , [ ((3, 13), (13, 3)) ]
  -- small and large crosses in 2 directions
  , [ ((5, 8), (11, 8)), ((8, 5), (8, 11)) ]
  , [ ((3, 8), (13, 8)), ((8, 3), (8, 13)) ]
  , [ ((5, 5), (11, 11)), ((5, 11), (11, 5)) ]
  , [ ((3, 3), (13, 13)), ((3, 13), (13, 3)) ]
  , [ ((3, 8), (13, 8)), ((8, 3), (8, 13)), ((3, 3), (13, 13)), ((3, 13), (13, ↙
      ↳ 3)) ]
  , [ ((5, 8), (11, 8)), ((8, 5), (8, 11)), ((5, 5), (11, 11)), ((5, 11), (11, ↙
      ↳ 5)) ]
  -- small and large triangles in 4 directions
  , loop [ (5, 5), (11, 5), (8, 11) ]
  , loop [ (3, 3), (13, 3), (8, 13) ]
  , loop [ (5, 11), (11, 11), (8, 5) ]
  , loop [ (3, 13), (13, 13), (8, 3) ]
  , loop (map swap [ (5, 5), (11, 5), (8, 11) ])
  , loop (map swap [ (3, 3), (13, 3), (8, 13) ])
  , loop (map swap [ (5, 11), (11, 11), (8, 5) ])
  , loop (map swap [ (3, 13), (13, 13), (8, 3) ])
  -- small and large squares in 2 directions
  , loop [ (5, 5), (11, 5), (11, 11), (5, 11) ]
  , loop [ (3, 3), (13, 3), (13, 13), (3, 13) ]
  , loop [ (5, 8), (8, 5), (11, 8), (8, 11) ]
  , loop [ (3, 8), (8, 3), (13, 8), (8, 13) ]
  ]
```

18 src/Pretty.hs

```
module Pretty (pretty) where

import Control.Monad (forM_, liftM2)
import Data.Array.MArray (thaw)
import Data.Array.ST (runSTUArray, writeArray, readArray)
import Data.Array.Unboxed (UArray, bounds)
import Data.Text (Text, pack)
import Data.List (sortBy)
```

```
import Data.Map (Map, toList)
import Data.Ord (comparing)

import Types

pretty :: Header -> UArray (Int, Int) Double -> Map Text Int -> (([Double], [Double]), ([Text], UArray (Int, Int) Double))
pretty header vals bs =
  let sticks = uncurry (ticks 20) (hSampleRange header)
      vticks = uncurry (ticks 20) (hValueRange header)
      labels = pack "(trace elements)" : (map fst . sortBy (comparing snd) . toList $ bs)
      coords = accumulate vals
  in  ((sticks, vticks), (labels, coords))

accumulate :: UArray (Int, Int) Double -> UArray (Int, Int) Double
accumulate a0 = runSTUArray $ do
  let ((b0,s0),(b1,s1)) = bounds a0
  a <- thaw a0
  forM_ [s0 .. s1] $ \s ->
    forM_ [b0 + 1 .. b1] $ \b ->
      writeArray a (b, s) =<< liftM2 (+) (readArray a (b - 1, s)) (readArray a (b, s))
  return a

ticks :: Int -> Double -> Double -> [Double]
ticks n mi ma =
  let k = nearestNice $ (ma - mi) / fromIntegral n
      m0 = fromIntegral (ceiling (mi / k) :: Integer) * k
      m1 = fromIntegral (floor   (ma / k) :: Integer) * k
  in  [m0, m0 + k .. m1 ]

nearestNice :: Double -> Double
nearestNice k0 = head . dropWhile (< k0) $ nices

nices :: [Double]
nices = [ f * k | f <- map (10**) [-6 ..], k <- [1,2,5] ]
```

19 src/Print.hs

```
module Print (print, printKey) where

import Prelude hiding (print)
import Data.Array.Unboxed (UArray, bounds, (!))
import Data.Text (Text, pack, unpack)
import Numeric (showFFloat)
import System.FilePath (replaceExtension)

import Types
import Graphics
import SVG (fillStyleName)

first :: (a -> c) -> (a, b) -> (c, b)
first f (a, b) = (f a, b)

filled0 :: Graphics -> Either PatternID RGB -> [Text] -> [Text]
filled0 gfx c = visual gfx (Just c) Nothing Nothing Nothing
```

```
print :: Graphics -> Bool -> Bool -> Bool -> Header -> [Double] -> [Double] -> [Text] -> UArray Int Double -> UArray (Int, Int) Double -> [Text]
print gfx notitle sepkey patterned header sticks vticks labels times coords =
  let bands = toPoints (bounds coords) times coords
      labels' = reverse labels
      filled = filled0 gfx
      (colours, defs)
        | patterned = first (map Left) . unzip $ map (pattern gfx) labels'
        | otherwise = (map (Right . colour) labels', [])
      polygons = concat . zipWith (\c ps -> filled c (polygon gfx ps)) colours . map (map p) $ bands
      key = concat . zipWith3 (keyBox (gW + border * 2.5) (border * 1.5) (gH / 16)) [(0::Int) ..] colours $ labels'
      keyBox x y0 dy i c l =
        let y = y0 + fromIntegral i * dy
        in  (filled c $ rect gfx (x, y + 0.1 * dy) (dy * 0.8, dy * 0.8)) ++
            text gfx Nothing Start 15 (x + dy, y + dy * 0.6) [l]
      w = 1280
      h = 720
      gW = (if sepkey then 1280 else 960) - 2 * border
      gH = if notitle then 720 - 2 * border else 720 - 3 * border
      border = 60
      textOffset = 10
      (xMin, xMax) = hSampleRange header
      (yMin, yMax) = hValueRange header
      gRange@((gx0,gy0),(gx1,gy1)) = ((border*1.5, gH + if notitle then border*0.5 else border*1.5), (gW + border*1.5, if notitle then border*0.5 else border*1.5))
      p = rescalePoint ((xMin, yMin), (xMax, yMax)) gRange
      title = text gfx Nothing Middle 25 (w / 2, border * 0.75) [hJob header, pack " (", hDate header, pack ")"]
      background = filled (Right white) $ rect gfx (0,0) (w,h)
      box = filled (Right white) $ rect gfx (gx0,gy1) (gW,gH)
      leftLabel = text gfx (Just (-90)) Middle 20 (border/2, (gy0 + gy1)/2) [hValueUnit header]
      leftTicks = concatMap (\(y,l) -> let { (x1, y1) = p (xMin, y) ; (x2, y2) = p (xMax, y) } in
          line gfx (x1 - border/2, y1) (x2, y2) ++
          if l then [] else text gfx Nothing End 15 (x1 - textOffset, y1 - textOffset) (showSI y)
        ) (zip vticks (replicate (length vticks - 1) False ++ [True]))
      bottomLabel = text gfx Nothing Middle 20 ((gx0 + gx1)/2, gy0 + border) [hSampleUnit header]
      bottomTicks = concatMap (\(x,l) -> let { (x1, y1) = p (x, yMin) ; (x2, y2) = p (x, yMax) } in
          line gfx (x1, y1 + border/2) (x2, y2) ++
          if l then [] else text gfx Nothing Start 15 (x1 + textOffset, y1+2*textOffset) (showSI x)
        ) (zip sticks (replicate (length sticks - 1) False ++ [True]))
  in  document gfx (w,h) (concat defs) . concat $
        [ background
        , visual gfx (Just (Right black)) Nothing (Just black) (Just 1) $ concat $
          (if notitle then [] else [title]) ++
          [ leftLabel
          , bottomLabel
```

```
            , leftTicks
            , bottomTicks
            , visual gfx Nothing (Just 0.7) Nothing Nothing $ concat $
                [ box
                , polygons
                ] ++ if sepkey then [] else [key]
            ]
        ]

printKey :: Graphics -> Bool {- ^ patterned -} -> Text -> (FilePath, [Text])
printKey gfx True label =
  let (pname, pdef) = pattern gfx label
  in  printKey' gfx pdef (Left pname) label
printKey gfx False label =
  let cname = colour label
  in  printKey' gfx [] (Right cname) label

printKey' :: Graphics -> [Text] -> Either PatternID RGB -> Text -> (FilePath, [Text])
printKey' gfx defs c label =
  ( replaceExtension (unpack (fillStyleName c)) "svg"
  , document gfx (boxSize, boxSize) defs (filled0 gfx c $ rect gfx (0, 0) (boxSize, boxSize))
  )

boxSize :: Double
boxSize = 27

toPoints :: ((Int,Int),(Int,Int)) -> UArray Int Double -> UArray (Int,Int) Double -> [[(Double,Double)]]
toPoints ((b0,s0),(b1,s1)) times coords =
  [ fwd ++ rwd
  | b <- [b1, b1 - 1 .. b0 + 1]
  , let fwd = [ (times ! s, coords ! (b - 1, s)) | s <- up b ]
  , let rwd = [ (times ! s, coords ! (b, s)) | s <- down b ]
  ]
  where
    -- ugly hack to force recomputation instead of sharing a large list
    up   b = [s0 + b - b .. s1]
    down b = [s1 + b - b, s1 - 1 .. s0]

showSI :: Double -> [Text]
showSI x | x < 1e3   = [ showF  x       ""  ]
         | x < 1e6   = [ showF (x/1e3 ) "k" ]
         | x < 1e9   = [ showF (x/1e6 ) "M" ]
         | x < 1e12  = [ showF (x/1e9 ) "G" ]
         | x < 1e15  = [ showF (x/1e12) "T" ]
         | otherwise = [ showF (x/1e15) "P" ]
  where
    showF y si = let (xs, ys) = break ('.' ==) $ showFFloat (Just 3) y ""
                     zs = reverse . dropWhile ('0' ==) . reverse . drop 1 $ ys
                 in  case zs of
                       "" -> pack $ xs ++ si
                       ws -> pack $ xs ++ "." ++ ws ++ si
```

20 src/Prune.hs

```
module Prune
  ( prune
  , Compare
  , cmpName
  , cmpSize
  , cmpStdDev
  ) where

import Data.Text (Text)
import Data.List (foldl', sortBy)
import Data.Ord (comparing)
import Data.Map.Strict (Map, toList, fromList)

type Compare a = a -> a -> Ordering

cmpName, cmpSize, cmpStdDev
  :: (Compare (Text, (Double, Double)), Compare (Text, (Double, Double)))
cmpName = (cmpNameAscending, cmpNameDescending)
cmpSize = (cmpSizeDescending, cmpSizeAscending)
cmpStdDev = (cmpStdDevDescending, cmpStdDevAscending)

cmpNameAscending, cmpNameDescending,
  cmpStdDevAscending, cmpStdDevDescending,
  cmpSizeAscending, cmpSizeDescending :: Compare (Text, (Double, Double))
cmpNameAscending = comparing fst
cmpNameDescending = flip cmpNameAscending
cmpStdDevAscending = comparing (snd . snd)
cmpStdDevDescending = flip cmpStdDevAscending
cmpSizeAscending = comparing (fst . snd)
cmpSizeDescending = flip cmpSizeAscending

prune :: Compare (Text, (Double, Double)) -> Double -> Int -> Map Text (Double, ↙
    ↳ Double) -> Map Text Int
prune cmp tracePercent nBands ts =
  let ccTotals = sortBy cmpSizeDescending (toList ts)
      sizes = map (fst . snd) ccTotals
      total = sum' sizes
      limit = (1 - tracePercent / 100) * total
      bigs = takeWhile (< limit) . scanl (+) 0 $ sizes
      bands = zipWith const ccTotals $ take nBands bigs
      ccs = map fst (sortBy cmp bands)
  in  fromList (reverse ccs `zip` [1 ..])

sum' :: [Double] -> Double
sum' = foldl' (+) 0
```

21 src/SVG.hs

```
{-# LANGUAGE OverloadedStrings #-}
module SVG (svg, fillStyleName) where

import Data.Text (Text, pack)
import qualified Data.Text as T
import Data.List (intersperse)
import Data.Monoid ((<>), mconcat)
import Numeric (showHex)
import Text.FShow.RealFloat (fshow, Double7(D7))
```

```
10
    import Graphics (Graphics(Graphics), Point, Size, Angle, FontSize, Anchor(..), ↙
        ↳ StrokeWidth, Opacity, RGB(..), PatternID, colour, patternIx)
    import qualified Graphics as G
    import Pattern (patterns)

15  svg :: Graphics
    svg =
      Graphics
      { G.text     = text
      , G.rect     = rect
20    , G.line     = line
      , G.polygon  = polygon
      , G.pattern  = pattern
      , G.visual   = visual
      , G.document = document
25    }

    text :: Maybe Angle -> Anchor -> FontSize -> Point -> [Text] -> [Text]
    text r a s (x,y) inner =
      let coords = case r of
30          Nothing -> ["x='", showF x, "' y='", showF y, "'"]
            Just ra -> ["transform='translate(" , showF x, "," , showF y, ") rotate↙
                ↳ (", showF ra, ")'"]
          showAnchor Start = "start"
          showAnchor Middle = "middle"
          showAnchor End = "end"
35    in  ["<text stroke='none' "] ++ coords ++ [" font-size='", showF s, "' text-↙
          ↳ anchor='", showAnchor a, "'>"] ++ map escape inner ++ ["</text>\n"]

    escape :: Text -> Text
    escape = T.concatMap escapeChar
      where
40      escapeChar '<' = "&lt;"
        escapeChar '>' = "&gt;"
        escapeChar '&' = "&amp;"
        escapeChar c   = T.singleton c

45  rect :: Point -> Size -> [Text]
    rect (x,y) (w,h) = ["<rect x='", showF x, "' y='", showF y, "' width='" , showF ↙
        ↳ w , "' height='" , showF h , "' />\n"]

    line :: Point -> Point -> [Text]
    line (x1,y1) (x2,y2) = ["<line x1='" , showF x1, "' x2='", showF x2, "' y1='", ↙
        ↳ showF y1, "' y2='", showF y2, "' />\n"]
50
    pattern :: Text -> (PatternID, [Text])
    pattern t = ("url(#" <> pid <> ")",
      [ "<pattern id='", pid, "' x='0' y='0' width='16' height='16' patternUnits='↙
          ↳ userSpaceOnUse'>\n"
      , "  <path fill='none' stroke-width='2' stroke='" , s, "' line-cap='round' d↙
          ↳ ='"] ++ intersperse " " (map p ps) ++ ["' />\n"
55    , "</pattern>\n"])
      where
        pid = mconcat ["P", pack (show ix), "p", T.tail s]
        s = showRGB rgb
        p ((x0, y0), (x1, y1)) = pack $ "M " ++ show x0 ++ "," ++ show y0 ++ " L " ↙
```

```
        ↪ ++ show x1 ++ "," ++ show y1
      rgb = colour t
      ix = patternIx t
      ps = patterns !! ix

visual :: Maybe (Either PatternID RGB) -> Maybe Opacity -> Maybe RGB -> Maybe ↙
    ↪ StrokeWidth -> [Text] -> [Text]
visual mfill mfillo mstroke mstrokew inner =
  let fill = maybe [] (\f -> [" fill='", either id showRGB f, "'"]) mfill
      fillo = maybe [] (\o -> [" fill-opacity='", showF o, "'"]) mfillo
      stroke = maybe [] (\s -> [" stroke='", showRGB s, "'"]) mstroke
      strokew = maybe [] (\w -> [" stroke-width='", showF w, "'"]) mstrokew
  in ["<g"] ++ fill ++ fillo ++ stroke ++ strokew ++ [">\n"] ++ inner ++ ["</g>\↙
      ↪ n"]

document :: Size -> [Text] -> [Text] -> [Text]
document (w,h) defs inner =
  [ "<?xml version='1.0' encoding='UTF-8' ?>\n"
  , "<svg xmlns='http://www.w3.org/2000/svg' version='1.0'"
  , " width='", showF w, "' height='", showF h, "'>\n"
  ] ++ ["<defs>\n"] ++ defs ++ ["</defs>\n"] ++ inner ++ ["</svg>\n"]

polygon :: [Point] -> [Text]
polygon ps = ["<path d='"] ++ path ps ++ ["' />"]

path :: [(Double,Double)] -> [Text]
path [] = error "SVG.path: empty"
path (p0:ps) =
  let lineTo p = [ " L " ] ++ showP p
  in  [ "M " ] ++ showP p0 ++ concatMap lineTo (ps ++ [p0]) ++ [ " Z" ]

showRGB :: RGB -> Text
showRGB (RGB r g b) =
  let toInt x = let y = floor (256 * x) in 0 `max` y `min` 255 :: Int
      hex2 i = (if i < 16 then ('0':) else id) (showHex i "")
  in  pack $ '#' : concatMap (hex2 . toInt) [r,g,b]

showP :: Point -> [Text]
showP (x,y) = [showF x, ",", showF y]

showF :: Double -> Text
showF x = pack $ fshow (D7 x)

fillStyleName :: Either PatternID RGB -> Text
fillStyleName (Left c) = T.drop 5 (T.init c)
fillStyleName (Right c) = T.singleton 'C' <> T.tail (showRGB c)
```

22 src/Total.hs

```
{-# LANGUAGE BangPatterns #-}
module Total (total) where

import Control.Monad.State.Strict (State(), execState, get, put)
import Data.List (foldl')
import Data.Map (Map, empty, lookup, insert, alter)
import Prelude hiding (init, lookup, lines, words, drop, length)
import Data.Text (Text, init, pack, unpack, lines, words, isPrefixOf, drop, ↙
```

```
    ↳ length)
import Data.Attoparsec.Text (parseOnly, double)

import Types

data Parse =
  Parse
  { symbols   :: !(Map Text Text) -- intern symbols to save RAM
  , totals    :: !(Map Text (Double, Double)) -- compute running totass and ↙
      ↳ total of squares
  , sampleMin :: !Double
  , sampleMax :: !Double
  , valueMin  :: !Double
  , valueMax  :: !Double
  , count     :: !Int                          -- number of frames
  }

parse0 :: Parse
parse0 = Parse{ symbols = empty, totals = empty, sampleMin = 0, sampleMax = 0, ↙
    ↳ valueMin = 0, valueMax = 0, count = 0 }

total :: Text -> (Header, Map Text (Double, Double))
total s =
  let ls = lines s
      (hs, ss) = splitAt 4 ls
      [job, date, smpU, valU] =
        zipWith header [sJOB, sDATE, sSAMPLE_UNIT, sVALUE_UNIT] hs
      parse1 = flip execState parse0 . mapM_ parseFrame . chunkSamples $ ss
  in  ( Header
        { hJob         = job
        , hDate        = date
        , hSampleUnit  = smpU
        , hValueUnit   = valU
        , hSampleRange= (sampleMin parse1, sampleMax parse1)
        , hValueRange = (valueMin parse1, valueMax parse1)
        , hCount       = count parse1
        }
      , fmap (stddev $ fromIntegral (count parse1)) (totals parse1)
      )

stddev :: Double -> (Double, Double) -> (Double, Double)
stddev s0 (s1, s2) = (s1, sqrt (s0 * s2 - s1 * s1) / s0)

header :: Text -> Text -> Text
header name h =
  if name `isPrefixOf` h
  then init . drop (length name + 2) $ h -- drop the name and the quotes
  else error $ "Parse.header: expected " ++ unpack name

chunkSamples :: [Text] -> [[Text]]
chunkSamples [] = []
chunkSamples (x:xs)
  | sBEGIN_SAMPLE `isPrefixOf` x =
      let (ys, zs) = break (sEND_SAMPLE `isPrefixOf`) xs
      in  case zs of
            [] -> [] -- discard incomplete sample
            (_:ws) -> (x:ys) : chunkSamples ws
```

```
    | otherwise = [] -- expected BEGIN_SAMPLE or EOF...

parseFrame :: [Text] -> State Parse ()
parseFrame [] = error "Parse.parseFrame: empty"
parseFrame (l:ls) = do
  let !time = sampleTime sBEGIN_SAMPLE l
  samples <- mapM inserter ls
  p <- get
  let v = foldl' (+) 0 samples
      sMin = if count p == 0 then time else time `min` sampleMin p
      sMax = if count p == 0 then time else time `max` sampleMax p
      vMin = v `min` valueMin p
      vMax = v `max` valueMax p
  put $! p{ count = count p + 1, sampleMin = sMin, sampleMax = sMax, valueMin = ↙
      ↳ vMin, valueMax = vMax }

inserter :: Text -> State Parse Double
inserter s = do
  let [k,vs] = words s
      !v = readDouble vs
  p <- get
  k' <- case lookup k (symbols p) of
    Nothing -> do
      put $! p{ symbols = insert k k (symbols p) }
      return k
    Just kk -> return kk
  p' <- get
  put $! p'{ totals = alter (accum  v) k' (totals p') }
  return $! v

accum :: Double -> Maybe (Double, Double) -> Maybe (Double, Double)
accum x Nothing  = Just $! (((,) $! x) $! (x * x))
accum x (Just (y, yy)) = Just $! (((,) $! (x + y)) $! (x * x + yy))

sampleTime :: Text -> Text -> Double
sampleTime name h =
  if name `isPrefixOf` h
  then readDouble .  drop (length name + 1) $ h
  else error $ "Parse.sampleTime: expected " ++ unpack name ++ " but got " ++ ↙
      ↳ unpack h

readDouble :: Text -> Double
readDouble s = case parseOnly double s of
  Right x -> x
  _ -> error $ "Parse.readDouble: no parse " ++ unpack s

sJOB, sDATE, sSAMPLE_UNIT, sVALUE_UNIT, sBEGIN_SAMPLE, sEND_SAMPLE :: Text
sJOB = pack "JOB"
sDATE = pack "DATE"
sSAMPLE_UNIT = pack "SAMPLE_UNIT"
sVALUE_UNIT = pack "VALUE_UNIT"
sBEGIN_SAMPLE = pack "BEGIN_SAMPLE"
sEND_SAMPLE = pack "END_SAMPLE"
```

23 src/Types.hs

```
module Types where
```

```
import Data.Text (Text)

data Header =
  Header
  { hJob         :: Text
  , hDate        :: Text
  , hSampleUnit  :: Text
  , hValueUnit   :: Text
  , hSampleRange :: (Double, Double)
  , hValueRange  :: (Double, Double)
  , hCount       :: Int
  }
```

24 tests/escapes.hs

```
module Main where

a <<< b = reverse a ++ reverse b
a &&& b = zipWith (\x y -> [x, y]) a b
foo = {-# SCC "&<>!'\"/" #-} id

main = interact main'
main' s = foo $ (s <<< concat (s &&& s)) <<< concat (s &&& s) <<< s
```

25 tests/namelen.hs

```
module Main where

a <<< b = {-# SCC "ThisSCCHasAnExtremelyLongNameThatMightBeChoppedOffPrematurely↙
    ↳ " #-} reverse a ++ reverse b
a &&& b = zipWith (\x y -> {-# SCC "↙
    ↳ ThisSCCHasAnotherExtremelyLongNameThatMightBeChoppedOffPrematurely" #-} [x↙
    ↳ , y]) a b
foo = id

main = interact main'
main' s = foo $ (s <<< concat (s &&& s)) <<< concat (s &&& s) <<< s
```

26 tests/unicode.hs

```
føb :: Integer -> Integer
føb n
  | n == 0 = 0
  | n == 1 = 1
  | n >= 2 = føb (n - 1) + føb (n - 2)

main :: IO ()
main = print (føb 100)
```

27 THANKS

```
Ian Lynagh (label text XML escaping bugfix patch)
Edward Z. Yang (feature requests for stable colours and uniform scales)
Pepe Iborra (parsing bugfix)
Hans-Peter Deifel (bug report about truncated key with long SCC names)
```