`xmonad` in Coq

Programming a window manager in a proof assistant

Wouter Swierstra
Haskell Symposium 2012

Coq

Coq

- Coq is ‘just’ a total functional language;
- *Extraction* lets you turn Coq programs into OCaml, Haskell, or Scheme code.
- Extraction discards proofs, but may introduce ‘unsafe’ coercions.

Demo

Extraction to Haskell
is not popular.

xmonad

xmonad

- A tiling window manager for X:
  - arranges windows over the whole screen;
  - written and configured in Haskell;
  - has several tens of thousands of users.

| |1|a-f|*[2]g-m|*[3]r-s|[4]t*|[5]boring/hints|[6]im|[7]mag*|[8]toggles|[9]winNav|[0]*| Thu Feb 26, 7:51 PM

File Edit Tools Syntax Buffers Window Help

```
----        ||| named "GridVariants.TallGrid 2 3 (2/3) (4/3) (5/100)" (GridV.TallGrid 2 3 (2/3) (4/3) (5/100))
            ||| named "HintedGrid.Grid False" (HGrid.Grid False)
            ||| named "HintedGrid.Grid True" (HGrid.Grid True)
            ||| named "HintedTile 2 (3/100) (1/2) TopLeft Tall" (HT.HintedTile 2 (3/100) (1/2) HT.TopLeft HT.Tall)
            ||| named "HintedTile 2 (3/100) (1/2) *Center* Tall" (HT.HintedTile 2 (3/100) (1/2) HT.Center HT.Tall)
            ||| named "HintedTile 2 (3/100) (1/2) *BottomRight* Tall" (HT.HintedTile 2 (3/100) (1/2) HT.BottomRight HT.Tall)
            ||| named "HintedTile 2 (3/100) (1/2) TopLeft Wide" (HT.HintedTile 2 (3/100) (1/2) HT.TopLeft HT.Wide)
            ||| named "HintedTile 2 (3/100) (1/2) Center Wide" (HT.HintedTile 2 (3/100) (1/2) HT.Center HT.Wide)
            ||| named "HintedTile 2 (3/100) (1/2) BottomRight Wide" (HT.HintedTile 2 (3/100) (1/2) HT.BottomRight HT.Wide)
            ||| named "mastered (3/100) (3/8) $ HintedGrid.Grid True" (mastered (3/100) (3/8) $ HGrid.Grid True)
            ||| named "mastered (1/118) (4/9) $ Mirror TwoPane"
                          (mastered (1/118) (4/9) $ Mirror $ TwoPane (3/100) (1/2))
            ||| named "MosaicAlt" (MosaicAlt M.empty)
            )
----        ||| named "Mirror Mosaic [19,5,3]" (Mirror (Mosaic [19,5,3]))
----        ||| named "Mosaic [12,8,5,3,2]" (Mosaic [12,8,5,3,2])
----        ||| named "Mosaic [4,7]" (Mosaic [4,7]) )

rsLayouts =   ResizableTall 2 (1/118) (11/20) [1]
            ||| named "Spiral (1/2) (default direction, orientation)" (spiral (1/2))
            ||| named "Spiral (golden ratio)" (spiral (toRational (2/(1+sqrt(5)::Double))))
            ||| named "StackTile 1 (3/100) (2/3)" (StackTile 1 (3/100) (2/3))

tLayouts   =  (named "Tall 1 (1/118) (5/9)" (XMonad.Tall 1 (1/118) (5/9))
            ||| named "ThreeCol 1 (1/118) (1/2)" (ThreeCol 1 (1/118) (1/2))
--            ||| named "ThreeColumnsMiddle 1 (1/118) (1/2)" (ThreeColMid 1 (1/118) (1/2))
            ||| named "TwoPane (3/100) (5/9)" (TwoPane (3/100) (5/9))
            ||| named "Mirror TwoPane (3/100) (2/3)" (Mirror $ TwoPane (3/100) (2/3)) )
            ||| simpleTabbed

borHintLayouts = boringWindows $
xmonad.hs [haskell][unix]                        211/416,85 52%
```

```
tmp
TODO
trash
wikidoc
xmonad.errors
xmonad.hi
xmonad.hs
xmonad.o
xmonad-x86_64-linux
vav@sibelius ~ $
```

```
SEEING WHAT YOU'VE DONE
       whatsnew
              whatsnew gives you a view of what changes you've made in your working copy that haven't
              yet been recorded. The changes are displayed in darcs patch format. Note that --look-
              for-adds implies --summary usage.

OTHER COMMANDS
       revert Revert is used to undo changes made to the working copy which have not yet been
              recorded. You will be prompted for which changes you wish to undo. The last revert can
              be undone safely using the unrevert command if the working copy was not modified in the
              meantime.

       unrevert
              Unrevert is used to undo the results of a revert command. It is only guaranteed to work
              properly if you haven't made any changes since the revert was performed.

       unrecord
              Unrecord does the opposite of record in that it makes the changes from patches active
              changes again which you may record or revert later. The working copy itself will not
              change.

       amend-record
Manual page darcs(1) line 78
```

File Edit View Go File Edit View Go

DARCS

Example of Use

Places ▾

vav

Desktop

Most Visited

Xmonad/Scr... gmane.com... Octree e...

Octree example

Via ffffound.

Would you like to comment?

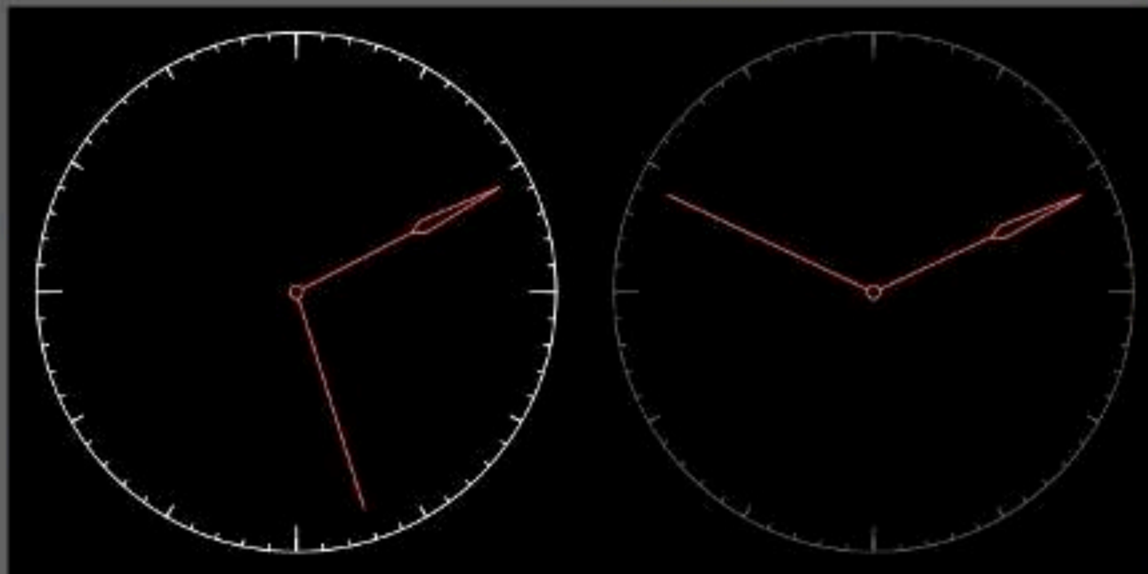
Sign up for a free account, or sign in (if you're already a member).

You Sign in | Create Your Free Account

Explore Places | Last 7 Days | This Month | Popular Tags | The Commons | Creati

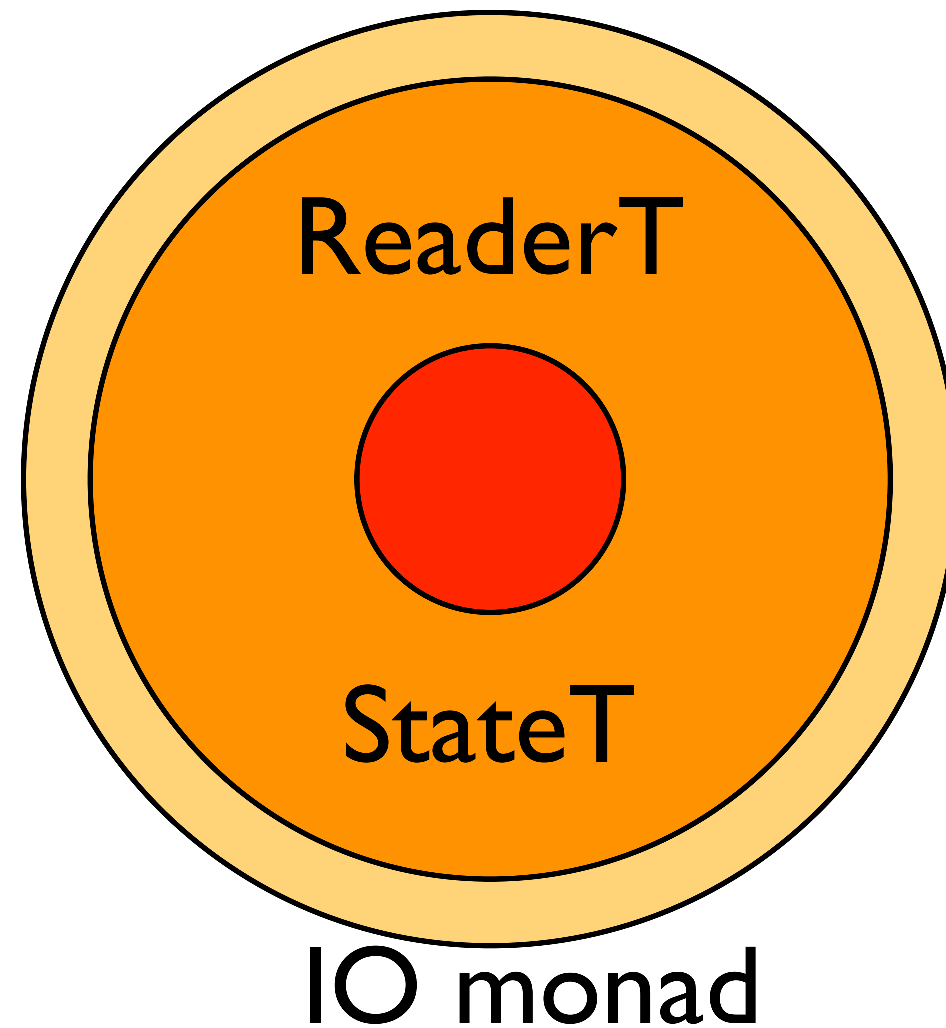
Help Community Guidelines | The Help Forum | FAQ | Sitemap | Help by Emai

http://www.flickr.com/photos/et [3/3] 90%



[Spiral (golden ratio)]

`xmonad`: design principles



This paper

- This paper describes a reimplementation of xmonad's pure core in Coq;
- Extracting Haskell code produces a drop-in replacement for the original module;
- Documents my experience.

Does it work?

Blood

Sweat

```
s/delete :: /delete :: Ord a3 => /g
s/remove0 :: /remove0 :: Ord a1 => /g
s/insert :: /insert :: Ord a1 => /g
s/sink :: /sink :: Ord a3 => /g
s/float ::/float :: Ord a3=> /g88d87
<         ghc-options:    -Werror
23c23
<      ScreenId(..), ScreenDetail(..), XState(..),
---
>      ScreenDetail(..), XState(..),
109c109
< type WindowSet   = StackSet  WorkspaceId (Layout Window) Window ScreenId Scree
nDetail
---
> type WindowSet   = StackSet  WorkspaceId (Layout Window) Window ScreenDetail
115,117d114
< -- | Physical screen indices
< newtype ScreenId    = S Int deriving (Eq,Ord,Show,Read,Enum,Num,Integral,Real)
<
131,132c131,132
<                       >>= W.filter (`M.notMember` W.floating ws)
<                       >>= W.filter (`notElem` vis)
```

Shell script

What happens in the functional core?

Data types

```
data Zipper a = Zipper
  { left :: [a]
  , focus :: !a
  , right :: [a]
  }
```

... and a lot of functions for zipper manipulation

Totality

- This project is feasible because most of the functions are structurally recursive.
- What about this function?

```
focusLeft (Zipper [] x rs) =
  let (y : ys) = reverse (x : rs)
  in Zipper ys y []
```

Extraction

- The basic extracted code is terrible!
  - uses Peano numbers, extracted Coq booleans, etc.
  - uses extracted Coq data types for zippers;
  - generates ‘non-idiomatic’ Haskell.

Customizing extraction

- There are various hooks to customize the extracted code:
  - inlining functions;
  - realizing axioms;
  - using Haskell data types.

Interfacing with Haskell

- We would like to use ‘real’ Haskell booleans

```
Extract Inductive bool =>
"Bool" ["True" "False" ].
```

- Lots of opportunity to shoot yourself in the foot!

Better extracted code

- The extracted file uses generated data types and exports ‘too much’
- *Solution*:
  - Customize extraction to use hand-coded data types.
  - Write a `sed` script that splices in a new module header and data type definitions.

Type classes

Type classes

- Haskell's function to check if an element occurs in a list:

```
elem :: Eq a => a -> [a] -> Bool.
```

- A Coq version might look like:

```
Variable a : Set.
Variable cmp : forall (x y : a),
  {x = y} + {x <> y}.
Definition elem : a -> list a -> ...
```

Extracted code

- Extracting this Coq code generates functions of type:

```
_elem :: (a -> a -> Bool) ->
   a -> [a] -> Bool.
```

- Need a manual ‘wrapper function’

```
elem :: Eq a => a -> [a] -> Bool
elem = _elem (==)
```

Further woes

- This doesn't scale well to 'bigger' type classes (like `Ord`, `Integral`, ...);
- Interfacing with existing libraries is an even greater pain;
- Additional `sed` scripts to postprocess the generated Haskell 'solve' these issues.

Result!

- This proves the core functions are total*
- Fixed a bug in `xmonad`.
- More than 25% QuickCheck properties formally verified in Coq.

https://github.com/wouter-swierstra/xmonad/

* under certain conditions.

Conclusions

- Formal verification can complement, but not replace a good test suite.
- Extraction can introduce bugs!
- If you want to do formal verification, but need `sed` to ‘fix’ your code, something is wrong...