Drive-by Haskell Contributions

Neil Mitchell

http://ndmitchell.com
Getting started contributing

• *Or*: ideas to improve your existing project
Goal: Start doing cooler stuff
Do: Check your change is welcome

- Is the project on GitHub?
- Look at the open PR’s – do they languish?
- When was the last commit?
- Does it compile with the latest deps?
- On Stackage?
- Improve things you use/believe-in
- Is there a contrib policy? Is it friendly?
- Ask before investing too much (github issue)
I welcome and appreciate contributions. If you've contributed to my code, and we meet in real life, I'll buy you a beer.

If you want to amend a pull request, rewrite your branch and leave a comment. Do not add commits to the branch or open new pull requests for that.
Don’t: Rearrange the deck chairs

e.g. Reindent, add -Wall, add new dependencies
Build and use it

• Was that easy?
• If not, improve the README
  – What it does
  – Why you should use it
  – How you use it (example)

• Maintainers have too much knowledge to do this well
Look at the Haddock

• Are the functions clear?
  – More examples required?
  – Are corner cases clear?
  – Add docs liberally, don’t worry about being wrong

• Haddock coverage stats are useless
  – I use “haddock --hoogle” then munge the output
Do the docs work

• Rule: Code that is not compiled rots
  – Includes Haddock comments and examples in the manual

• Manually check a few instances
  – Report any buggy examples
  – Perhaps a bigger project of automatic checks?
Is the bug tracker clean?

• Are all the things on the bug tracker still relevant?
• Are there things on the bug tracker that are related but not cross-linked?
• Beware: Don’t want to add to maintainer woes
Apply static checkers

• Do: apply static checkers, report good finds
• Don’t: make maintainers use them

• Maintainers may choose to use the static checker if the payoff is high, but that’s up to them
-Wall

• cabal build --ghc-options=-Wall
  – Get a list of the issues, which make sense?
• Example: Shake has 895 warnings
  – Most in the test suite, plenty unused arguments

Shake\Classes.hs:5:15: warning: [-Wdodgy-exports]
The export item `Typeable(..)' suggests that `Typeable' has methods, but it has none

A thread to pull on, not an answer
HLint

- `cabal install hlint && hlint . --report`
  - See `report.html`, which make sense?

---

Report generated by `HLint` v0.0 - a tool to suggest improvements to your Haskell code.

Sample.hs:5:7: Warning: Use and
Found
foldr1 (&&)
Why not
and

Note: removes error on []
HLint best hints

• HLint reports a lot – find the “good stuff”

• From Shake:

```haskell
{-# LANGUAGE GeneralizedNewtypeDeriving, DeriveDataTypeable, ScopedTypeVariables, ConstraintKinds #-}
{-# LANGUAGE UndecidableInstances, TypeFamilies, ConstraintKinds #-}
```
HLint good stuff

- Redundant language extensions
- Use of mapM instead of mapM_
- Simple sugar functions (concatMap)
  - Look for refactor introduced noise

- Don’t rearrange the deck chairs:
  - If vs case
  - Redundant lambda
How HLint works

• Parse the source (using haskell-src-exts)
• Traverse the syntax tree (using uniplate)
• Some hints are hardcoded (e.g. extensions)
• Most hints are expression templates
  – \{\text{lhs: map (uncurry f) (zip x y)}, \text{rhs: zipWith f x y}\}
  – \{\text{lhs: not (elem x y), rhs: notElem x y}\}
  – \{\text{lhs: any id, rhs: or}\}
How HLint works

findIdeas :: [HintRule] -> Scope ->
-> Decl_ -> [Idea]

findIdeas matches s decl =
[ (idea (hintRuleSeverity m) (hintRuleName m) x y
[ r]){ideaNote=notes}
| (parent,x) <- universeParentExp decl, not $ isPare n x,
, m <- matches, Just (y,notes, subst, rule) <- [matchIdea s
decl m parent x]
, let r = R.Replace R.Expr (toSS x) subst (prettyPrint rule)]
Weeder

• Finds the “weeds” in a program — weeder.

= Package ghcid

== Section exe:ghcid test:ghcid_test
Module reused between components
  * Ghcid
    Weeds exported
  * Wait
    - withWaiterPoll

Module used in two cabal projects

Function exported but not used elsewhere

http://hackage.haskell.org/package/weeder
Weeder best hints

• Code is exported and not used outside
  – Delete the export

• GHC warnings detect definition is unused
  – Delete the code entirely

• Package dependency is not used
  – Remove a dependency (see also packdeps)
How Weeder works

• Stack compiles with dump .hi files
  – Each module has a large blob of text
• Parse these .hi files, extract relevant data
  – What packages you make use of
  – What imported identifiers you use
• Analyse
  – If ‘foo’ is exported, but not used, it’s a weed
How Weeder works

data Hi = Hi
  {hiModuleName :: ModuleName
   -- ^ Module name
  ,hilImportPackage :: Set.HashSet PackageName
   -- ^ Packages imported by this module
  ,hiExportIdent :: Set.HashSet Ident
   -- ^ Identifiers exported by this module
  ,hilImportIdent :: Set.HashSet Ident
   -- ^ Identifiers used by this module
  ,hilImportModule :: Set HashSet ModuleName
   -- ^ Modules imported and used by this module

HLInt and Weeder

- Both have binary releases on github

  curl -sL https://.../hlint/travis.sh | sh -s .

- Both have ignore files

  weeder . --yaml > .weeder.yaml
  hlint . --default > .hlint.yaml
Tests are great

- Writing good tests takes time – often missed
- Find tests that are missing
  - Will often lead to bugs (also fun to fix)

- Beware, tests are not always good:
  - Verbosity (don’t check a dumb 1 liner)
  - Performance (1M iterations of QuickCheck)
  - Maintenance (do they need updating often)
Read bug reports

• Take a bug report
  – Is there a reproducible case? If not, write it
  – Is the test case machine checked? If not, make it
  – Is it ready to go in the test suite? If not, make it

• Now you have your test
  – Is it fixed? Great, submit a pull request with it
  – Is it still broken? Share the test anyway
Use HPC

- Run the test suite through HPC
  ghc -fhpc Main.hs && ./main
  hpc report main.tix && hpc markup main.tix

readRule :: HintRule -> [HintRule]
readRule (m@HintRule{hintRuleLHS=(fmapAn -> hintRuleLHS), hintRu
  (:) m{hintRuleLHS=hintRuleLHS,hintRuleSide=hintRuleSide,hint
    (l,v1) <- dotVersion hintRuleLHS
    (r,v2) <- dotVersion hintRuleRHS
    guard $ v1 == v2 && l /= [] && (length l > 1 || length r
    if r /= [] then
      [m{hintRuleLHS=dotApps l, hintRuleRHS=dotApps r, hin
      ,m{hintRuleLHS=dotApps (l++[toNamed v1]), hintRuleRH
      else if length l > 1 then
        [m{hintRuleLHS=dotApps l, hintRuleRHS=toNamed "id",
        ,m{hintRuleLHS=dotApps (l++[toNamed v1]), hintRuleRH
      else []}]}]}]}]
HPC – complex and untested

• Do: Look for the sweet spot
  – Code that is not obviously correct
  – Code that is untested
  – Add a test based on its docs (are they sufficient?)

• Don’t: Aim for 100% coverage
  – You want to reach that, not aim for it
  – Incentives matter
Run on Travis/Appveyor

• A good CI is important for a project
  – Travis = Linux/Mac, Appveyor = Windows

• Very time consuming to set up

• There is a lot of variety
  – Hvr provides a PPA archive of GHC binaries
  – Stack can grab GHC binaries
  – I use bootstrap scripts
Bootstrap scripts

• Each repo...
curl -sL https://.../travis.sh | sh

• ...calls a centralised shell script...
apt-get install ghc-$GHCVER
cabal install neil
./neil

• ...which calls Haskell
system_ "cabal check"

No \r
Installs cleanly
Full documentation
Lowercase cabal keys
Performance

• It’s nice for most code to be faster, smaller
  – But make sure the tests are reasonable first

• Do: Check performance matters
  – Saving 20% on a 1ms operation is often useless
  – Saving 50% on something running yearly is useless
  – All these apply to memory as well
The simple view

- Measure, Whack, repeat
  - Something to measure
  - Somehow to direct your whack
Time profiling

ghc Main.hs -prof -auto-all && ./Main +RTS -p

- HLint generates 6590 lines, top is a table

<table>
<thead>
<tr>
<th>COST CENTRE</th>
<th>MODULE</th>
<th>% TIME</th>
<th>% ALLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>unifyExp</td>
<td>Hint.Match</td>
<td>23.0</td>
<td>2.0</td>
</tr>
<tr>
<td>findIdeas</td>
<td>Hint.Match</td>
<td>10.5</td>
<td>0.2</td>
</tr>
<tr>
<td>uniplateData</td>
<td>Data.Generics.Uniplate.Internal.Data</td>
<td>7.5</td>
<td>19.8</td>
</tr>
<tr>
<td>matchIdea</td>
<td>Hint.Match</td>
<td>6.1</td>
<td>12.8</td>
</tr>
<tr>
<td>follower</td>
<td>Data.Generics.Uniplate.Internal.Data</td>
<td>4.1</td>
<td>1.0</td>
</tr>
<tr>
<td>pushContextL</td>
<td>Language.Haskell.Exts.ParseMonad</td>
<td>4.0</td>
<td>5.3</td>
</tr>
</tbody>
</table>
# Time profiling tree

<table>
<thead>
<tr>
<th>COST CENTRE</th>
<th>MODULE</th>
<th>ENTRIES</th>
<th>%TIME</th>
<th>%ALLOC</th>
<th>%TIME</th>
<th>%ALLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>unifyExp</td>
<td>Hint.Match</td>
<td>16744713</td>
<td>23</td>
<td>2</td>
<td>29.3</td>
<td>11.8</td>
</tr>
<tr>
<td>isDot</td>
<td>HSE.Util</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>rebracket</td>
<td>Hint.Match</td>
<td>41</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>opExp</td>
<td>HSE.Util</td>
<td>152744</td>
<td>0</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>nmOp</td>
<td>Hint.Match</td>
<td>678224</td>
<td>0</td>
<td>0</td>
<td>1.7</td>
<td>3.4</td>
</tr>
<tr>
<td>isDol</td>
<td>HSE.Util</td>
<td>706356</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>matchIdea.nm</td>
<td>Hint.Match</td>
<td>831226</td>
<td>0</td>
<td>0</td>
<td>1.8</td>
<td>2.5</td>
</tr>
<tr>
<td>fromParen</td>
<td>HSE.Util</td>
<td>433761</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>fromNamed</td>
<td>HSE.Match</td>
<td>1728163</td>
<td>0.2</td>
<td>0</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>isUnifyVar</td>
<td>Config.Type</td>
<td>1728163</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

In reality, way harder to view...
Time profiling - Profiteur

http://hackage.haskell.org/package/profiteur
Time profiling - Profiterole

- Profiterole generates 442 lines, CSE and roots

<table>
<thead>
<tr>
<th>TOT</th>
<th>INH</th>
<th>IND</th>
<th>Time (ms)</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>51.0</td>
<td>47.4</td>
<td></td>
<td>51.0</td>
<td>Hint.Match readMatch (53)</td>
</tr>
<tr>
<td>12.0</td>
<td>12.0</td>
<td></td>
<td>12.0</td>
<td>Data.Generics.Uniplate.Internal.Data readCacheFollower (3)</td>
</tr>
<tr>
<td>10.3</td>
<td>10.2</td>
<td></td>
<td>10.2</td>
<td>Language.Haskell.Exts parseFileContentsWithComments (53)</td>
</tr>
<tr>
<td>8.7</td>
<td>7.5</td>
<td>7.5</td>
<td>8.7</td>
<td>Data.Generics.Uniplate.Internal.Data uniplateData (1377837)</td>
</tr>
<tr>
<td>99.9</td>
<td>5.2</td>
<td></td>
<td>5.2</td>
<td>MAIN MAIN (0)</td>
</tr>
<tr>
<td>2.9</td>
<td>2.8</td>
<td>2.0</td>
<td>2.9</td>
<td>Data.Generics.Uniplate.Internal.Data descendBiData (109203)</td>
</tr>
<tr>
<td>2.4</td>
<td>2.4</td>
<td></td>
<td>2.4</td>
<td>HSE.All runCpp (53)</td>
</tr>
</tbody>
</table>
Profiterole tower

<table>
<thead>
<tr>
<th>TOT</th>
<th>INH</th>
<th>IND</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.0</td>
<td>12.0</td>
<td></td>
<td>Data.Generics.Uniplate.Internal.Data readCacheFollower (3)</td>
</tr>
<tr>
<td>7.2</td>
<td>7.2</td>
<td></td>
<td>Data.Generics.Uniplate.Internal.Data insertHitMap (2)</td>
</tr>
<tr>
<td>7.2</td>
<td>7.2</td>
<td>6.2</td>
<td>Data.Generics.Uniplate.Internal.Data fixEq (7)</td>
</tr>
<tr>
<td>7.2</td>
<td>7.2</td>
<td></td>
<td>Data.Generics.Uniplate.Internal.Data set_unions (0)</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>Data.HashMap.Array new_ (558259)</td>
</tr>
<tr>
<td>4.8</td>
<td>4.8</td>
<td>4.1</td>
<td>Data.Generics.Uniplate.Internal.Data follower (2)</td>
</tr>
<tr>
<td>.5</td>
<td>.5</td>
<td>.5</td>
<td>Data.HashMap.Base sparseIndex (635260)</td>
</tr>
</tbody>
</table>

Previously readCacheFollower was in 155 distinct places
How Profiterole works

- Read GHC .prof with ghc-prof library
- Build a Tree Val, Val = \{Name, TOT, INH, IND\}
- Find roots
  - Called by more than 2 places, or in a config file
- Lift roots to the top-level
- Merge equally named roots
- Write back out
- Can take 200K lines to 5K
Memory profiling

ghc Main.hs -prof -auto-all && ./Main +RTS -hm
hp2ps -c Main
Stack profiling

ghc --make Main.hs -rtsopts -prof -auto-all

- Compile with profiling

./Main +RTS -K${N}K

- Find lowest ${N} where program works

./Main +RTS -xc -K${N-1}K

- Get a stack trace, examine it

- Fix. Repeat until -K1K works

Find performance bugs in vector, base, QuickCheck, happy, pretty...
Let the drive-by contributions begin!