API Annotations

enabler for source code round tripping / modification
Background
Major issues

● Managing change
  ○ add / delete lines
  ○ changing indentation
  ○ preserving layout

● No separation of essential and accidental code
  ○ fiddly token management
Solution: get rid of tokens

- Approach taken by exactprint in haskell-src-exts
- But this is also brittle and fiddly
  - Annotated locations are present, but changing anything requires changing everything
  - Basically same change problems as with tokens
- But still a step in the right direction
Concept for HaRe

- Use GHC AST updated as per HSE
- Convert the fixed locations into relative
  - Essentially convert to pretty printer directives
  - A HaRe intermediate step had been haskell-token-utils, did something similar using dual-tree from diagrams
GHC - 7.8 - Plan

- landmines in AST
- add API Annotations
  - basically the equivalent of HSE Annotated
- capture all original source literals
GHC 7.10.2

- 40 commits over 10 months
- But, able to round-trip most of hackage
Landmines - PlaceHolder.hs

- | Types that are not defined until after type checking

```hs
type family PostTc it ty :: * -- Note [Pass sensitive types]

-- Note [Pass sensitive types]
type instance PostTc Id ty = ty

-- Note [Pass sensitive types]
type instance PostTc Name ty = PlaceHolder

-- Note [Pass sensitive types]
type instance PostTc RdrName ty = PlaceHolder
```

-- | Types that are not defined until after renaming

```hs

-- Note [Pass sensitive types]
type family PostRn id ty :: * -- Note [Pass sensitive types]

-- Note [Pass sensitive types]
type instance PostRn Id ty = ty

-- Note [Pass sensitive types]
type instance PostRn Name ty = ty

-- Note [Pass sensitive types]
type instance PostRn RdrName ty = PlaceHolder
```
API Annotations

data AnnKeywordId
    = AnnAs
    | AnnAt
    | AnnBang  -- ^ '!' 

....

type ApiAnnKey = (SrcSpan,AnnKeywordId)
type ApiAnns = ( Map.Map ApiAnnKey [SrcSpan]
                 , Map.Map SrcSpan [Located AnnotationComment])
Parser

| 'do' stmtlist

{% ams (L (comb2 $1 $2))
  (mkHsDo DoExpr (snd $ unLoc $2)))
  (mj AnnDo $1:(fst $ unLoc $2))
%

-- |Add a list of AddAnns to the given AST element
ams :: Located a -> [AddAnn] -> P (Located a)
ams a@(L l _) bs = mapM_ (\a -> a l) bs >> return a

-- |Construct an AddAnn from the annotation keyword and the location
-- of the keyword
mj :: AnnKeywordId -> Located e -> AddAnn
mj a l = (\s -> addAnnotation s a (gl l))
Example

2: foo = do

3: let x = 1 -- a comment

4: return x

([((tests/examples/SimpleDo.hs:(2,1)-(4,10), AnnEqual), [tests/examples/SimpleDo.hs:2:5]),
  ((tests/examples/SimpleDo.hs:(2,1)-(4,10), AnnFunId), [tests/examples/SimpleDo.hs:2:1-3]),
  ((tests/examples/SimpleDo.hs:(2,1)-(4,10), AnnSemi), [tests/examples/SimpleDo.hs:5:1]),
  ((tests/examples/SimpleDo.hs:(2,7)-(4,10), AnnDo), [tests/examples/SimpleDo.hs:2:7-8]),
  ((tests/examples/SimpleDo.hs:3:3-14, AnnLet), [tests/examples/SimpleDo.hs:3:3-5]),
  ((tests/examples/SimpleDo.hs:3:3-14, AnnSemi), [tests/examples/SimpleDo.hs:4:3]),
  ((tests/examples/SimpleDo.hs:3:10-14, AnnEqual), [tests/examples/SimpleDo.hs:3:12]),
  ((tests/examples/SimpleDo.hs:3:10-14, AnnFunId), [tests/examples/SimpleDo.hs:3:10]),
  (<no location info>, AnnEofPos),
  [(tests/examples/SimpleDo.hs:2,7)-(4,10) [AnnLineComment "-- a comment"] ]})
ghc-exactprint

- inspired by haskell-src-exts exactprint
- but with changes driven by HaRe
  - Must allow changes to the AST
  - Fully local edit operations, and not dependent on SrcSpan
  - Automatically manage layout rules
- modelled on pretty-printer
- separate library from HaRe
ghc-exactprint phases

- **Delta** - relativise annotations
- **Transform** - manipulate AST
- **Print** - recreate original source, with changes
ghc-exactprint annotations

data KeywordId = G GHC.AnnKeywordId
  | AnnSemiSep
  | AnnComment Comment
  | AnnString String
  | AnnUnicode GHC.AnnKeywordId
deriving (Eq,Ord)

data AnnKey = AnnKey GHC.SrcSpan AnnConName
  deriving (Eq,Ord)

type Anns = Map.Map AnnKey Annotation
data Annotation = Ann

  { annEntryDelta :: DeltaPos
  , annPriorComments :: [(Comment, DeltaPos)]
  , annFollowingComments :: [(Comment, DeltaPos)]

  , annsDP :: [(KeywordId, DeltaPos)]
  , annSortKey :: (Maybe [GHC.SrcSpan])
  , annCapturedSpan :: (Maybe AnnKey)

} deriving (Typeable, Eq)
```hs
2:   foo = do
3:     let x = 1 -- a comment
4:     return x

{ tests/examples/SimpleDo.hs:2:7-(4,10) }
Just (Ann (DP (0,1)) [] [] [(G AnnDo),DP (0,0)]) Nothing Nothing)

(HsDo
(DoExpr)
[

{ tests/examples/SimpleDo.hs:3:3-14 }
Just (Ann (DP (1,2)) [] [] [(G AnnLet),DP (0,0)]) Just [tests/examples/SimpleDo.hs:3:10-14] Nothing)

(LetStmt
(HsValBinds
(ValBindsIn {Bag(Located (HsBind RdrName))}):
[

{ tests/examples/SimpleDo.hs:3:10-14 }
Just (Ann (DP (0,4)) [] [] Nothing Nothing)

(FunBind
.....
```
foo  xxx = let a = 1
        b = 2 in  xxx + a + b

foo xxxlonger = let a = 1
               b = 2 in  xxxlonger + a + b
ghc-exactprint Transform

- Transform monad
- manages annotations and new SrcSpans
  - SrcSpan AnnConName is only an index into anns, can freely add or remove SrcSpans
- Provides operations to simplify modifications
HasDecls

class `(Data t) => HasDecls t` where

hsDecls :: t -> Transform [GHC.LHsDecl GHC.RdrName]

replaceDecls :: t -> [GHC.LHsDecl GHC.RdrName] -> Transform t

```
-- |This is a function
foo = x -- comment1
```

```
-- |This is a function
foo = x -- comment1
   where
     nn = 2
```

class `(Monad m) => (HasTransform m)` where

`liftT :: Transform a -> m a`
module RmDecl2 where

sumSquares x y = let sq 0=0
                sq z=z^pow
                pow=2
                in sq x + sq y

anotherFun 0 y = sq y
             where sq x = x^2

doRmDecl lp = do
         let go :: GHC.LHsExpr GHC.RdrName -> Transform (GHC.LHsExpr GHC.RdrName)
            go e@(GHC.L_ (GHC.HsLet {})) = do
decs <- hsDecls e
    e' <- replaceDecls e (init decs)
    return e'
    go x = return x

SYB.everywhereM (SYB.mkM go) lp
Identity Transformation

- A source to source tool is useless if it cannot do the identity transformation
- Matthew Pickering results for hackage
  - 50,000 files successfully roundtripped (excl CPP)
  - 40 failures well-categorised and being attended to for GHC 7.12
    - CPP file end on Mac / Clang [not checked]
    - multi-line string literals in pragmas
    - unicode *
Apply-refact

- GSOC project to apply hlint hints via ghc-exactprint (Matthew Pickering)
- Successful outcome - demo
- Validates ghc-exactprint approach
Next steps

- More AST cleanups for ParsedSource
  - Make sure every RdrName is Located
- Investigate keeping information in RenamedSource AST
- OR providing lookup table from Located RdrName to Name
- More GHC API support for tool makers
References

- [https://github.com/alanz/ghc-exactprint](https://github.com/alanz/ghc-exactprint)
- [https://github.com/alanz/HaRe](https://github.com/alanz/HaRe)
- [https://github.com/mpickering/apply-refact](https://github.com/mpickering/apply-refact)
Questions?