Look Ma, No Signatures!
Separate Modular Development without Interfaces

Edward Z. Yang
Look Ma, No Signatures!

How to compile mutually recursive modules without hs-boot files

Edward Z. Yang
The problem...

#1409 new feature request

Allow recursively dependent modules transparently (without .hs-boot or anything)

 Reported by: Isaac Dupree
 Priority: normal
 Component: Compiler
 Keywords: backpack

 Owned by: 
 Milestone: 
 Version: 6.10.2

 Cc: dterei, pho@cielonegro.org, alfonso.acosta@gmail.com, mnsilah@gmail.com, lazycat.manatee@gmail.com, alexander.dunlap@gmail.com, id@isaac.cedarswampstudios.org, lennart@augustsson.net, ben.franksen@online.de, nr@cs.tufts.edu, ndmitchell@gmail.com, asr, explicitcall@gmail.com, kolmodin@gentoo.org, lambda-belka@yandex.ru, analytic@gmail.com, tora@zonetora.co.uk, sveina@gmail.com, ganesh, illissius@gmail.com, hackage.haskell.org@liyang.hu, skilpat@mpi-sws.org
The problem...

“Manually keeping hs-boot files... is a nuisance.”

“It's not a little problem.”

“I absolutely hate it if I have to write a [.hs-boot] file.”

“I waste too much time [fixing] import cycle[s].”
The problem?

Clearly a lot of people are interested in this ticket, but I'm not clear why:

- Writing an hs-boot file is very like writing a module signature in ML; and that in turn is a bit like writing a type signature on a function. Why is that so bad?

- At the moment I am very un-clear about the design of a viable alternative. Lennart's recent suggestion comes closest; but does that meet the goals of others.

Once there is a clear motivation, and a clear design, it'll become easier to comment about how easy or difficult it is to implement.

Simon

"The ML community considers it a major virtue that module signatures and implementations are separate."
Two design paths

Compile SCC as one monolithic unit

Automatically infer hs-boot file
Two design paths

- Compile SCC as one monolithic unit
  - Check: Best performing generated code
  - X: No recompilation avoidance
  - X: How does ghc -c work?
  - !: Unclear impact on GHC's architecture
Two design paths

✓ Recompilation avoidance and one-shot work

✗ Cannot inline across hs-boot

✗ How do you infer the hs-boot file?

Automatically infer hs-boot file
Two design paths

- Recompilation avoidance and one-shot work
- Cannot inline across hs-boot

How do you infer the hs-boot file?

Automatically infer hs-boot file
Trouble:

module A where
  import B
  data A = MkA B
            fromB :: B → A
            fromB (MkB a) = a

module B where
  import A
  data B = MkB A
            fromA :: A → B
            fromA (MkA b) = b

assume explicit type signatures
module A where
  import B

data A = MkA B

fromB :: B → A

fromB (MkB a) = a
Trouble:

module A where
  import B
  data A = MkA B
  fromB :: B → A
  fromB (MkB a) = a

module B where
  -- B.hs-boot
  fromB (MkB a) = a
  data B = MkB A
module A where
  import B
  data A = MkA B
  fromB :: B -> A
  fromB (MkB a) = a

-- \hs-boot\nmodule B where
  data B = MkB A

hs-boot files may need to import hs-boot files
so how do we do it?
so how do we do it?

Stratification
module A where
  import B
  data A = MkA B
  fromB :: B → A
  fromB (MkB a) = a
hs-boot

data types
value type signatures

hs
value implementations
data types

defines

references

types
values
constructors

types
hs-boot

module A where
  import {-#SOURCE2#} B
  data A = MkA B
  fromB :: B → A
  fromB (MkB a) = a

module B where
  import {-#SOURCE2#} A
  data B = MkB A
  fromA :: A → B
  fromA (MkA b) = b

hs-boot2

module A where
  data A

module B where
  data B
module A where
  data A = MkA B
  fromB :: B → A

module B where
  import {-#SOURCE2#} A
  data B = MkB A
  fromA :: A → B

module A where
  import {-#SOURCE#} B
  data A = MkA B
  fromB :: B → A
  fromB (MkB a) = a

module B where
  import {-#SOURCE#} A
  data B = MkB A
  fromA :: A → B
  fromA (MkB b) = b
The plan

1. Break import cycles with `{-#SOURCE #-}`

2. If an hs-boot file is absent: in memory
   - Generate an hs-boot file by erasing value implementations
   - Generate an hs-boot2 file for each module in the SCC with only abstract types
The plan

1. Break import cycles with `{{-#SOURCE#-}}`

2. If an hs-boot file is absent:
   - Generate an hs-boot file by erasing value implementations
   - Generate an hs-boot2 file for each module in the SCC with only abstract types

Imports in SCC are `{{-#SOURCE2#-}}`, others are normal
Implications for Backpack

Every implementation implicitly defines a signature
Implications for Backpack

```plaintext
unit  myapp where
    include mylib (A)
module  App where
    import A
```

(Why yes, they are named units now. And the inline module syntax does work!)
Implications for Backpack

```haskell
unit myapp where
  signature A where
    foo :: Int → Int

module App where
  import A
```

ugh, an hs-boot file!
Implications for Backpack

```
unit myapp where
  require mylib (A)
  module App where
    import A

infer the required interface
```
Implications for Backpack

- If you don’t use a value/type, it’s not counted for your requirement

- hs-boot2 files don’t influence type identity
Implications for Backpack

- Existing packages are Backpack packages

name: network
build-depends: base, bytestring, unix

unit network where
require base
require bytestring
require unix
Warning: Merging types with kinds

data A

data B :: A → *

data C :: B a → *

data D :: C b → *
Warning: Merging types with kinds

data A

data B :: A \rightarrow *

data C :: B a \rightarrow *

data D :: C b \rightarrow *

hs-boot4
hs-boot3
hs-boot2
hs-boot
Warning: Merging types with kinds

data A

data B :: A → *

data C :: B a → *

data D :: C b → *

hs-boot⁴
hs-boot³
hs-boot²
hs-boot

Possible fix: explicitly mark the "max level"
Look Ma, More Signatures!

defines...

hs-boot2  abstract types  requires...
            (nothing)

hs-boot  value type signatures  types
         data types

hs  value implementations  values
    data types
            constructors
            types

Thank you!