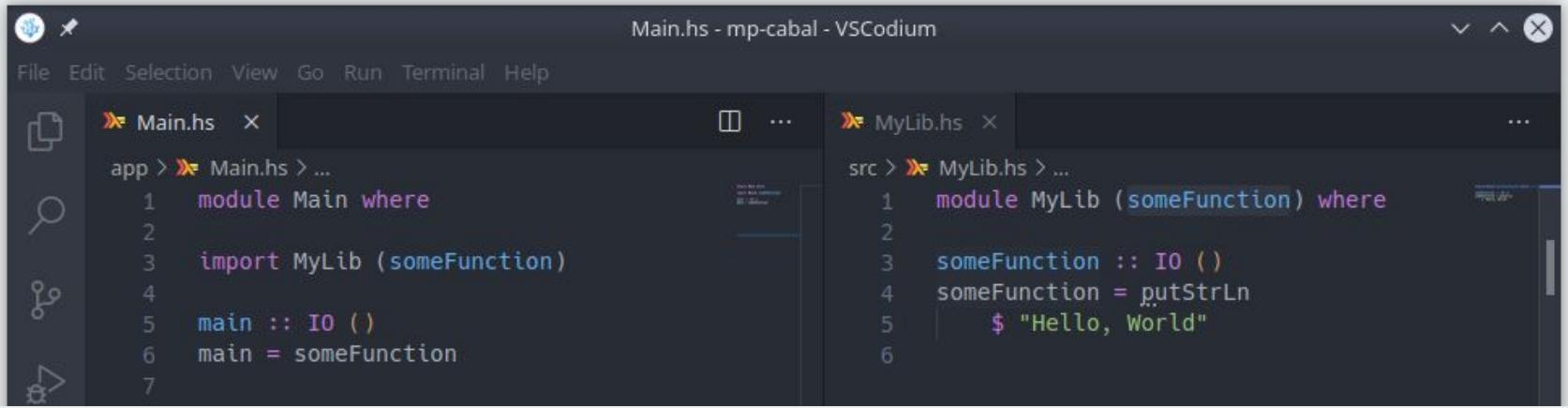
Multiple Home Units for GHC

Hannes Siebenhandl • 28.08.2020

Haskell IDE Engine



Haskell IDE Engine

Main.hs - mp-cabal - VSCodium

File Edit Selection View Go Run Terminal Help

Main.hs

app > Main.hs > ...

```haskell
module Main where

import MyLib (someFunction)

main :: IO ()
main = someFunction
```

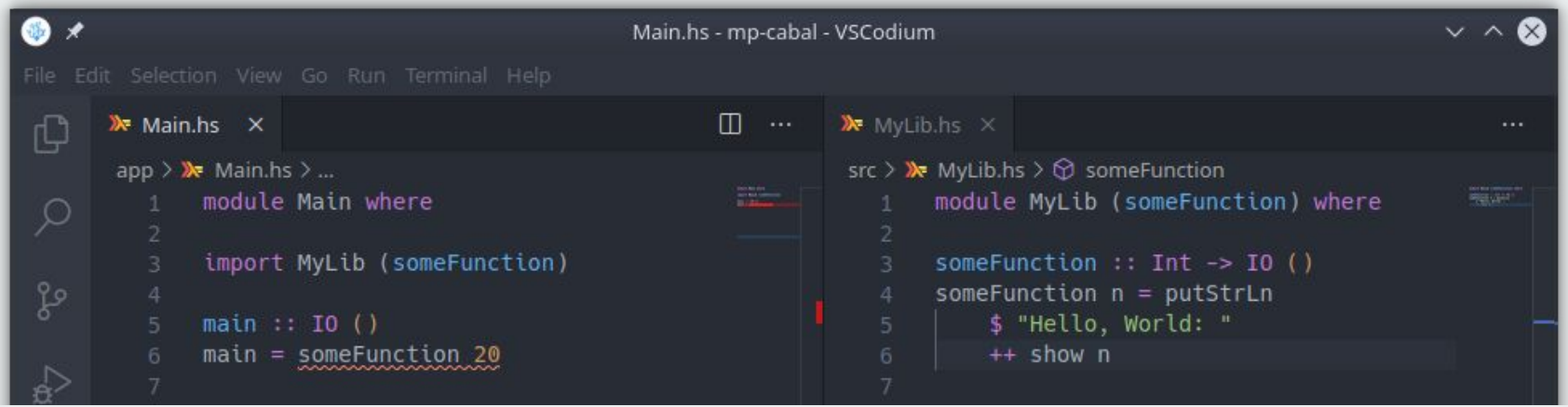
MyLib.hs

src > MyLib.hs > ...

```haskell
module MyLib (someFunction) where

someFunction :: Int -> IO ()
someFunction n = putStrLn
    $ "Hello, World: "
    ++ show n
```

Haskell IDE Engine



Cabal

```
> cabal repl lib:mp-cabal
Build profile: -w ghc-8.8.3 -O1
In order, the following will be built (use -v for more details):
 - mp-cabal-0.1.0.0 (lib) (ephemeral targets)
Preprocessing library for mp-cabal-0.1.0.0..
GHCi, version 8.8.3: https://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/munin/.ghci
[1 of 1] Compiling MyLib            ( src/MyLib.hs, interpreted )
Ok, one module loaded.
*MyLib
λ>
```

```
> cabal repl exe:mp-cabal
Build profile: -w ghc-8.8.3 -O1
In order, the following will be built (use -v for more details):
 - mp-cabal-0.1.0.0 (exe:mp-cabal) (first run)
Preprocessing executable 'mp-cabal' for mp-cabal-0.1.0.0..
GHCi, version 8.8.3: https://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/munin/.ghci
[1 of 2] Compiling Main             ( app/Main.hs, interpreted )
[2 of 2] Compiling Other            ( app/Other.hs, interpreted )
Ok, two modules loaded.
*Main
λ>
```

```
> cabal repl lib:mp-cabal exe:mp-cabal
cabal: Cannot open a repl for multiple components at once. The targets
'mp-cabal' and 'mp-cabal' refer to different components..

The reason for this limitation is that current versions of ghci do not support
loading multiple components as source. Load just one component and when you
make changes to a dependent component then quit and reload.
```

Stack

```
> stack repl
Using main module: 1. Package `simple-stack' component simple-stack:exe:simple-stack-exe with
/simple-stack/app/Main.hs
The following GHC options are incompatible with GHCi and have not been passed to it: -threade
Configuring GHCi with the following packages: simple-stack
GHCi, version 8.8.3: https://www.haskell.org/ghc/  :? for help
Loaded GHCi configuration from /home/munin/.ghci
[1 of 2] Compiling Lib              ( /home/munin/Documents/haskell/simple-stack/src/Lib.hs,
[2 of 2] Compiling Main             ( /home/munin/Documents/haskell/simple-stack/app/Main.hs,
Ok, two modules loaded.
Loaded GHCi configuration from /run/user/1000/haskell-stack-ghci/5e7f6527/ghci-script
*Main Lib
λ>
```

```
> bat /run/user/1000/haskell-stack-ghci/5e7f6527/ghci-script
File: /run/user/1000/haskell-stack-ghci/5e7f6527/ghci-script
1   :add Lib /home/munin/Documents/haskell/simple-stack/app/Main.hs
2   :module + Lib
>
```

All of these issues have a common cause!

Home Unit

What is a Home Unit?

- Consists of a set of modules to compile
- Describes how to compile those

- There is only one Home Unit

Currently

```
data HscEnv
 = HscEnv {

        hsc_dflags :: DynFlags,

        hsc_HPT :: HomePackageTable,

        ...
        }
```

- Single set of compilation options
- Single table for home modules

❌ Handle modules with the same name

❌ Single set of dependencies

With Multiple Home Units

```
data HscEnv
 = HscEnv {

       hsc_internalUnitEnv :: UnitEnv,

       …

 }

type UnitEnv = UnitEnvGraph InternalUnitEnv
```

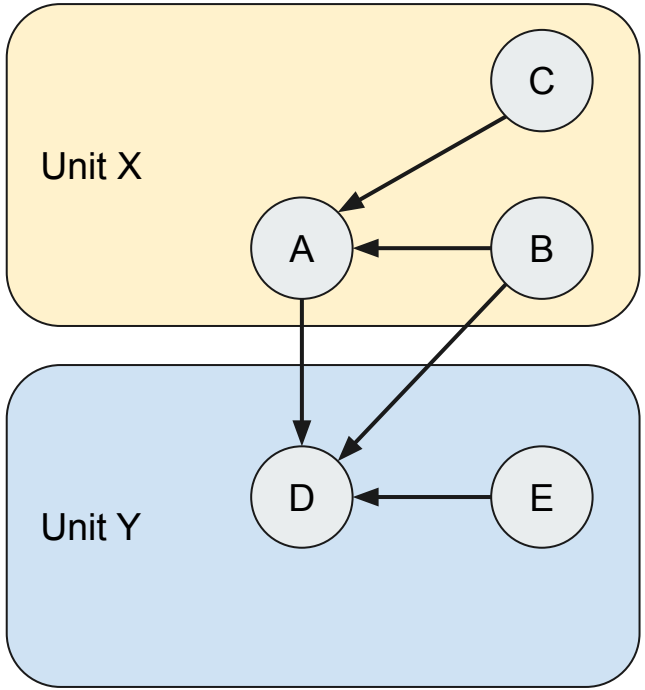
```
data UnitEnvGraph v = UnitEnvGraph
 { unitEnv_graph :: !(Map UnitId v)
 , unitEnv_currentUnit :: !UnitId
 }


data InternalUnitEnv = InternalUnitEnv
 { internalUnitEnv_dflags :: DynFlags
 , internalUnitEnv_homePackageTable
                          :: HomePackageTable
 }
```

Features

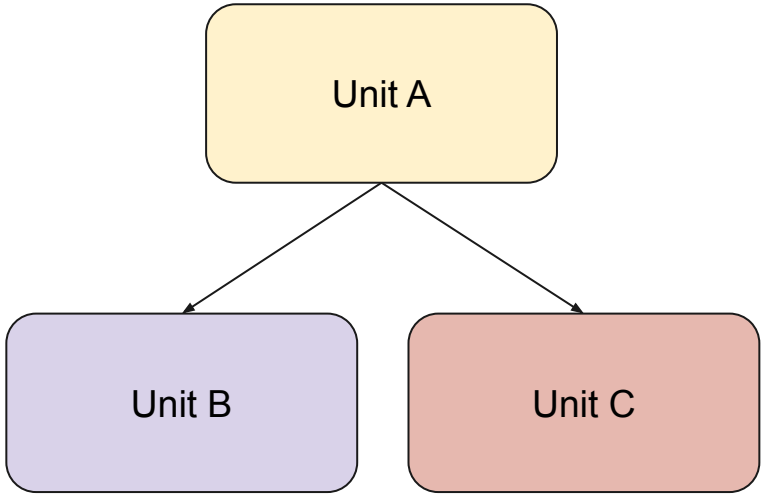
Downsweep / Upsweep

- Obtain module graph across all units
- Compile each module with the appropriate options



Home Unit Dependencies

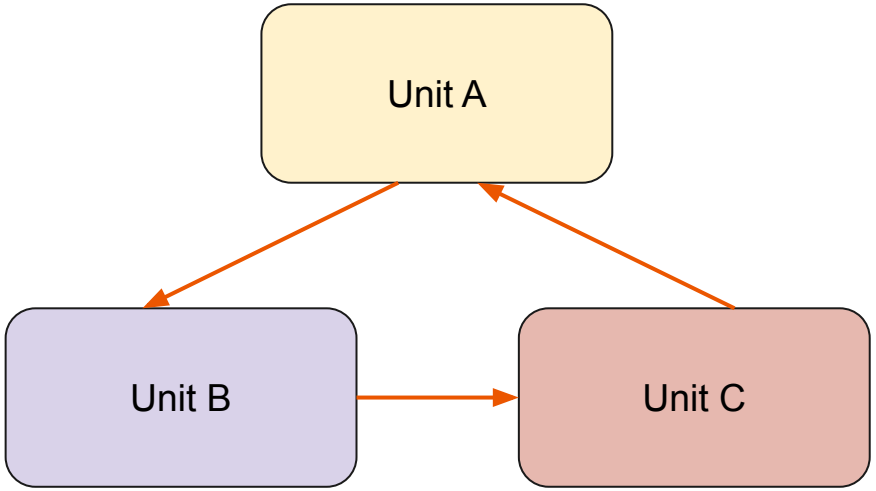
- Options for Unit B:
  `… -package base …`
- Options for Unit C:
  `… -package base …`
- Options for Unit A:
  `-package-id unitB` `-package-id unitC`



Home Unit Dependencies

- Must handle home unit dependencies differently

- Additional cycle detection required



GHC CLI

```
ghc --interactive -unit @unitA -unit @unitB … -unit @unitZ
```

The new mode uses response files for specifying compilation arguments for each unit.

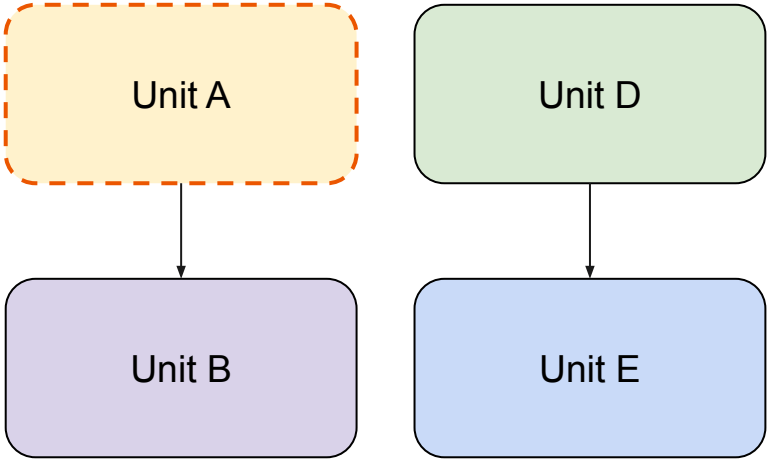
GHCi UI

- `:setunit <unit-id> <options>*`
  Set options for the given UnitId
- `:addunit <unit-id> <targets>*`
  Add targets for a specific unit
- `:switch <unit-id>`
  Switch currently active “main” unit.

GHCi UI

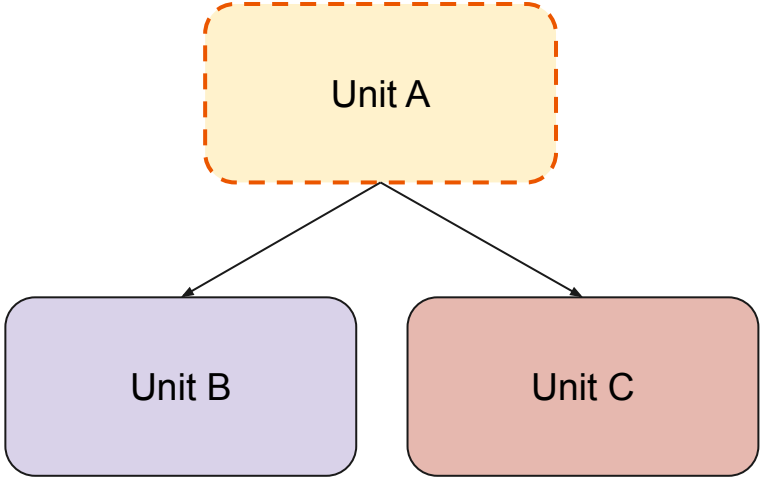
Why is this statefulness required?

- Avoids ambiguity when an identifier is used defined in both `UnitA` and `UnitD`



GHCi UI

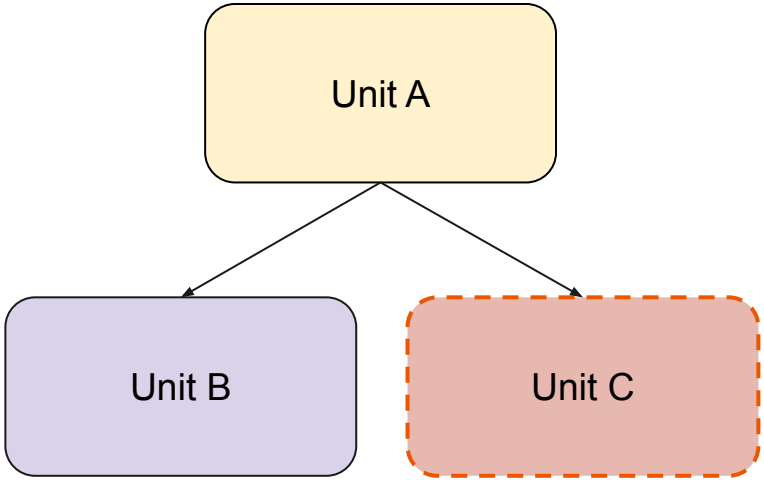
- `UnitA` is currently active
- Dependencies of `UnitA` are in scope
- Changes to `UnitB` or `UnitC` are propagated to `UnitA`



GHCi UI

After executing `:switch UnitC`

- `UnitC` is currently active
- Dependencies of `UnitC` are in scope
- Functions from `UnitA` can not be invoked



Live Demo

Future Work

Integrate into GHC

It has not been reviewed, but it is time now!

Make Tools use our Feature

Lift the limitations in tools such as cabal and stack.

Integrate into IDEs

Questions?

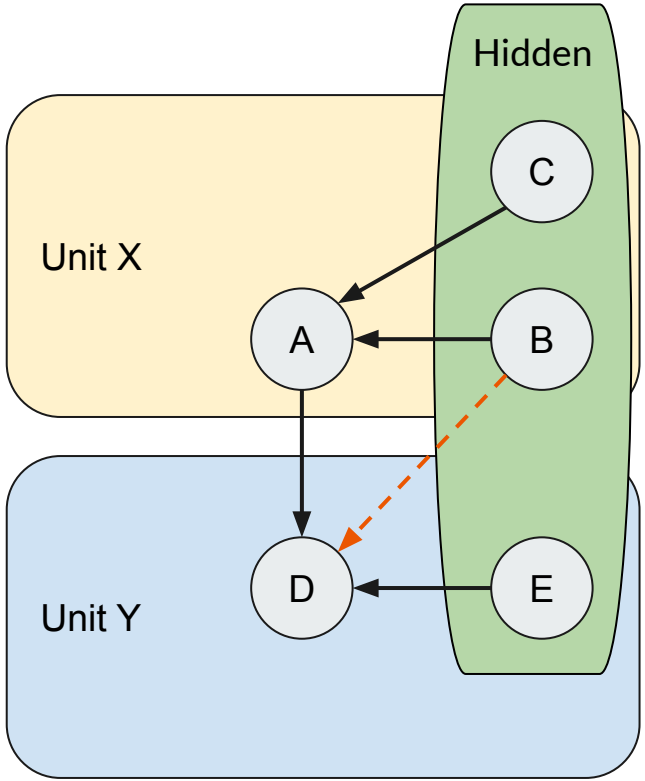
Limitations

Module Visibility

Module visibility depends on the package specification.

- No way to specify the visibility!

- Compilation succeeds although D depends on hidden module B.



Package Imports

- Dependencies are specified as:
  … `-package-id unitB-<hash>` …

- No way to get package name from UnitId

```
import "unitB" Foo

import "unitC" Foo
```