Генерация экземпляров классов типов на основе экземпляров производных классов в языке Haskell

О. Е. Филиппская

Направление подготовки: Прикладная математика и информатика
Руководитель: асс. каф. ИВЭ А. М. Пеленицын

Южный федеральный университет
Институт математики, механики и компьютерных наук имени И. И. Воровича

Кафедра информатики и вычислительного эксперимента

Цель работы

Проблема

- Applicative Monad Proposal (AMP) вносит предложение сделать класс Monad подклассом класса Applicative.
- GHC 7.8 $\longrightarrow$ GHC 7.10 — реализует AMP.
- Нарушена обратная совместимость.

Цель

- Создать программное средство, которое восстановит компилируемость программ.

Постановка задачи

1. Получить синтаксическое дерево программы.
2. Отфильтровать узлы, соответствующие экземплярам класса **Monad**.
3. Изменить нужным образом экземпляр класса **Monad** и добавить, если требуется, экземпляры классов **Applicative** и **Functor**.
4. Сформировать новый файл с текстом программы с помощью функций структурной печати (*pretty-printing*).

Метод решения

- GHC API — программный интерфейс компилятора

Пример

Пользовательский тип

```
newtype Prob a =
    Prob { getProb ::
           [(a, Rational)] }
```

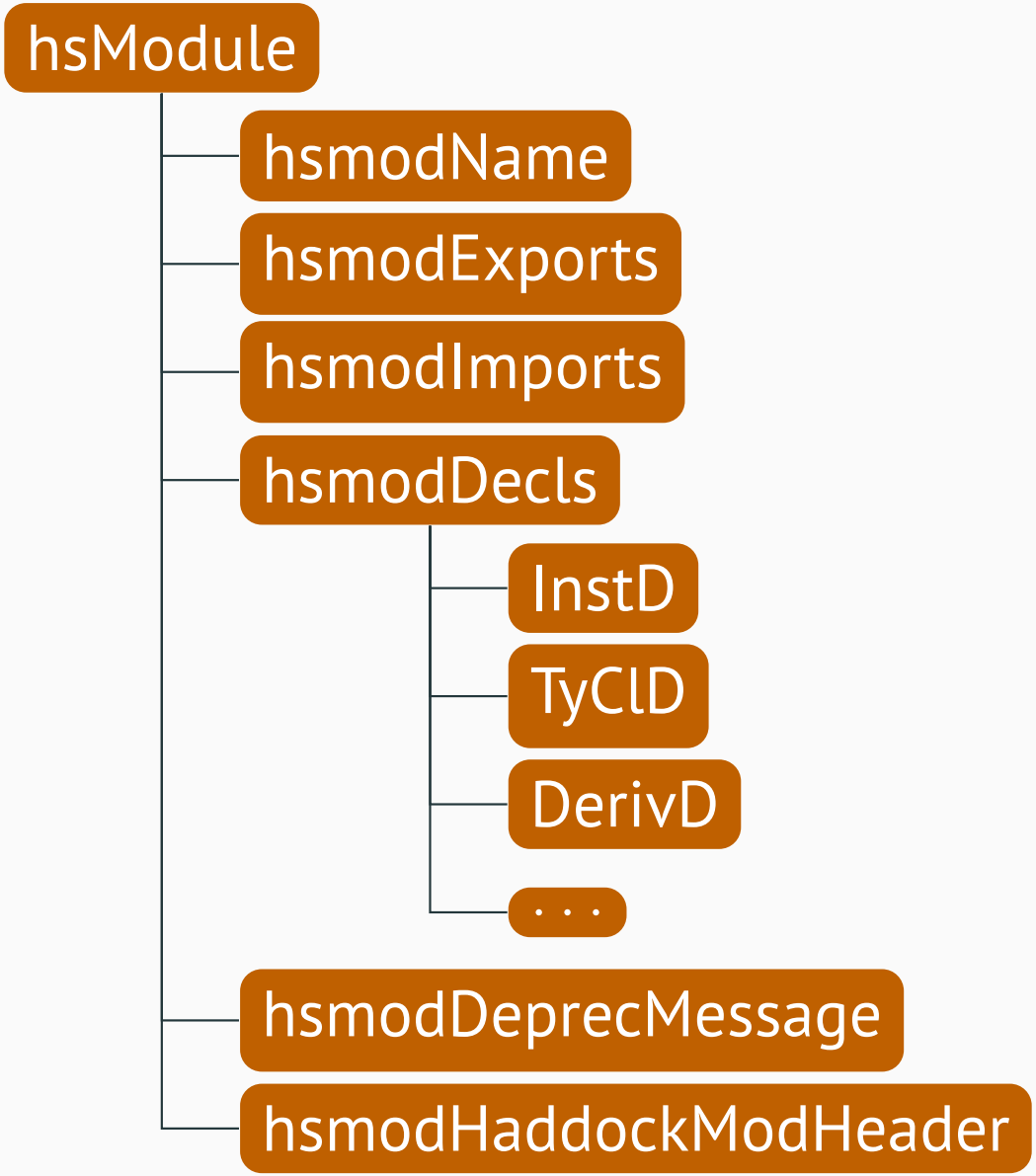
Экземпляр Monad

```
instance Monad Prob where
  return x = Prob [(x,1%1)]
  m >>= f  = flatten
                (fmap f m)
  x >> y   = x >>= \_ -> y
```

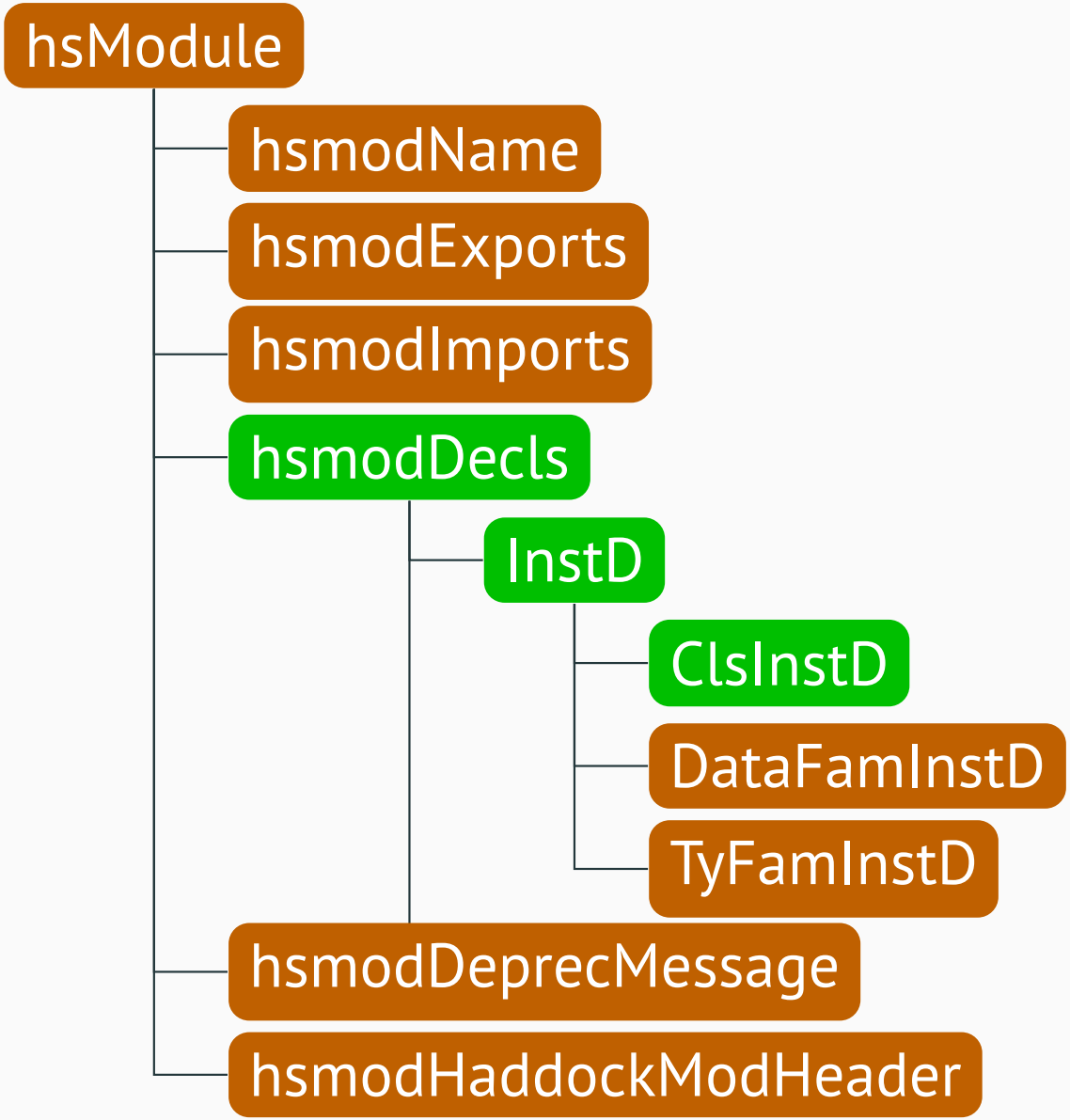
Результат

```
instance Functor Prob where
  fmap   = liftM

instance Applicative Prob
  where
  pure   = Prob [(x,1%1)]
  (<*>)  = ap
  x *> y = x >>= \_ -> y

instance Monad Probe where
  return   = pure
  (>>)     = (*>)
  m >>= f  = flatten
                (fmap f m)
```

Задача 1: получить синтаксическое дерево



Задача 2: отфильтровать экземпляры Monad



Задача 3: генерация экземпляров Applicative

One.hs

```haskell
module One where

import Control.Monad

data MyData a = MyData a
instance Monad MyData
```

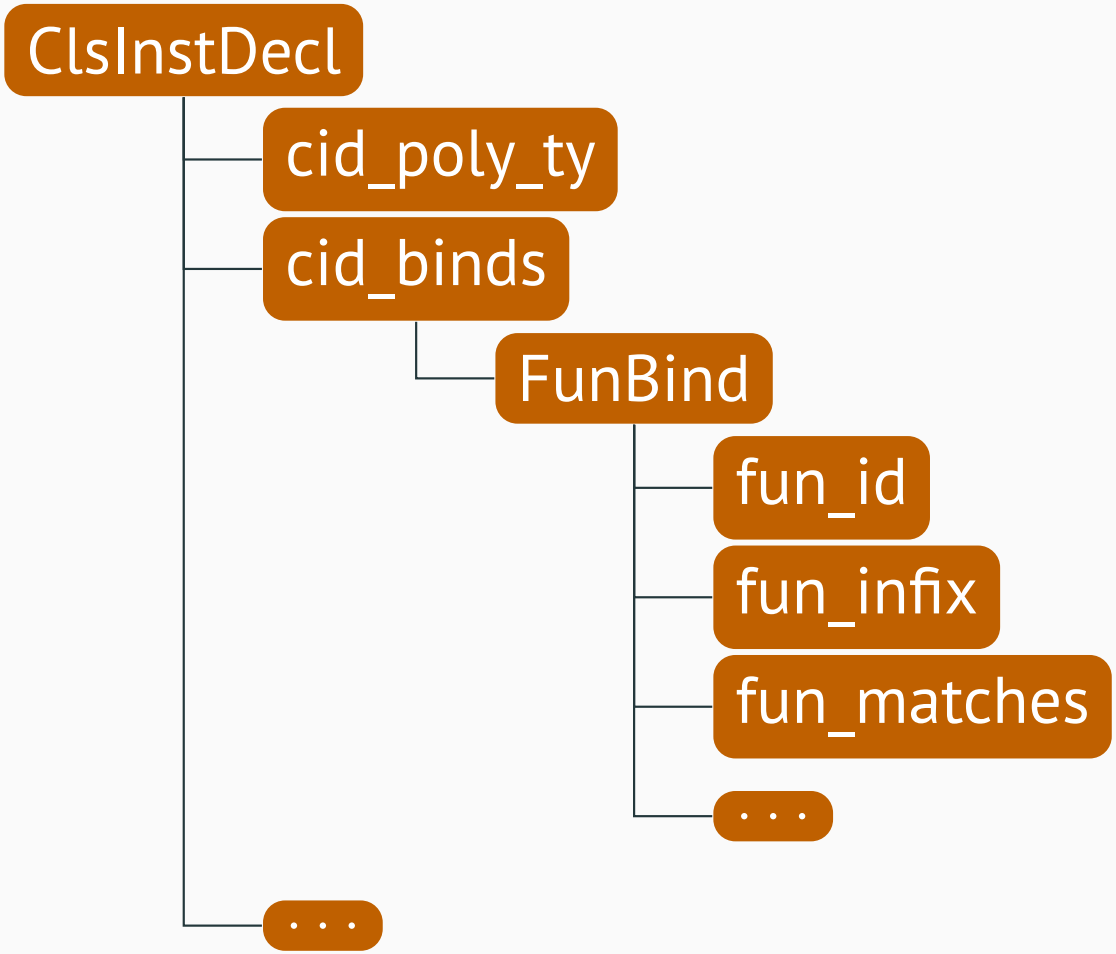
Two.hs

```haskell
module Two where

import Control.Applicative
import Data.Functor
import One

instance Functor MyData
instance Applicative MyData
```

Ошибка:

```
One.hs:7:10:
  No instance for (Applicative MyData)
<...>
```

Задача 3: генерация экземпляров Applicative



Более подробно:

https://ghc.haskell.org/trac/ghc/wiki/Migration/7.10

Задача 4: формирование выходного файла

- **Outputable** — класс типов, описанный в GHC API, экземпляры которого поддерживают функции структурной печати (pretty-printing).
- **HsModule** имеет экземпляр класса **Outputable**.
- ppr :: Outputable a **=>** a –> SDoc

Недостатки

1. Комментарии в выходном файле не сохраняются.

2. Конструкции **case** .. **of** на выходе синтаксически неверны (ошибка в исходном коде компилятора).

Синтаксическая ошибка:

```
f mp = case mp of {
           Nothing -> 1  -- missing ;
           Just _ -> 2 }
```

Результаты работы

1. Получено синтаксическое дерево программы (**parseModule**).
2. Полученное дерево проанализировано на наличие экземпляров класса **Monad** (**ClsInstDecl**).
3. Изменены нужным образом экземпляр класса **Monad** и добавлены экземпляры классов **Applicative** и **Functor**.
4. Сформирован новый файл с текстом программы с помощью функций структурной печати (**ppr**).

Репозиторий с исходными кодами:

- https://github.com/Valoisa/Generate-ancestor-instances