free-theorems: Syntaxbaum

Thomas Rossow

September 8, 2016

1 BasicSyntax

Das Paket free-theorems nutzt intern die selbstdefinierte Datenstruktur BasicSyntax als vereinfachte Version eines Syntaxbaumes zu einem Haskellprogramm. Diese Datenstruktur beinhaltet lediglich die Konstrukte, die für die Generierung von freien Theoremen benötigt werden: Von Interesse sind hier hauptsächlich natürlich Funktionssignaturen, daneben aber auch mit `data` definierte Datentypen und - insbesondere für die Erweiterung - Klassendefinitionen und Klasseneinschränkungen. Intern nutzt das Paket die Parser aus Language.Haskell.Exts und transformiert den resultierenden Syntaxbaum in die BasicSyntax.

1.1 Funktionssignaturen

Das wichtigste Element sind Funktionssignaturen. Bei der Überführung in die BasicSyntax werden sämtliche Funktionsimplementationen ignoriert und lediglich die Signaturen übernommen.

```
test :: a -> b

[TypeSig
  (Signature {
    signatureName =
      Ident {
        unpackIdent = "test" },
    signatureType =
      TypeFun
        (TypeVar
          (TV
            (Ident {
              unpackIdent = "a"}
            )
          )
        )
        (TypeVar
          (TV
            (Ident {
              unpackIdent = "b"}
            )
```

```
          )
        )}
  )
]
```

1.2 Eigene Datentypen

Da auch Datentypen in Funktionssignaturen vorkommen können, müssen die Datentypdeklarationen ebenfalls vorliegen. Zu beachten ist hier, dass die Typen der jeweiligen Datenkonstruktoren in `BangTypeExpression` verpackt sind: Der Konstruktor `Banged` wird verwendet, wenn der entsprechende Typ als "strikt" deklariert ist, ansonsten wird `Unbanged` benutzt.

```
data Test a = Test String a

[DataDecl
  (Data {
    dataName =
      Ident {
        unpackIdent = "Test"},
    dataVars = [
      TV (Ident {unpackIdent = "a"})
    ],
    dataCons = [
      DataCon {
        dataConName = Ident {unpackIdent = "Test"},
        dataConTypes = [
        Unbanged {
          withoutBang =
            TypeCon (Con (Ident {unpackIdent = "String"})) []},
        Unbanged {
          withoutBang =
            TypeVar (TV (Ident {unpackIdent = "a"}))}
        ]}
    ]}
  )
]
```

1.3 Eigene Typklassen

Besonders wichtig für die Erweiteurng von free-theorems sind die Typklassen. Auch diese werden in die Deklarationsliste mit aufgenommen, wobei auch hier die tatsächlichen Implementierungen der Funktionen keine Rolle spielen, lediglich die Signaturen werden verwendet.

```
class Monad m where
  (>>=) :: m a -> (a -> m b) -> m b

test :: Monad m => m a -> m a
```

```
[ClassDecl
  (Class {
    superClasses = [],
    className = Ident {unpackIdent = "Monad"},
    classVar = TV (Ident {unpackIdent = "m"}),
    classFuns = [
      Signature {...}
    ]}
  ),
  TypeSig
    (Signature {
      signatureName =
        Ident {unpackIdent = "test"},
        signatureType =
          TypeAbs
            (TV (Ident {unpackIdent = "m"}))
            [TC (Ident {unpackIdent = "Monad"})]
            (TypeFun
              (TypeVarApp
                (TV (Ident {unpackIdent = "m"}))
                [TypeVar (TV (Ident {unpackIdent = "a"}))]
              )
              (TypeVarApp
                (TV (Ident {unpackIdent = "m"}))
                [TypeVar (TV (Ident {unpackIdent = "a"}))]
              )
            )}
    )
]
```

Zu sehen ist zum einen die Deklaration einer Klasse sowie einer Funktion `test`, deren Typvariable `m` auf die Klasse Monad eingeschränkt ist. `classFuns` enthält eine Liste von Deklarationen in dieser Klasse. Für dieses Beispiel ergibt das die folgende Datenstruktur:

```
Signature {
  signatureName =
    Ident {unpackIdent = "(>>=)"},
    signatureType =
      TypeFun
        (TypeVarApp
          (TV (Ident {unpackIdent = "m"}))
          [TypeVar (TV (Ident {unpackIdent = "a"}))]
        )
        (TypeFun
          (TypeFun (TypeVar (TV (Ident {unpackIdent = "a"}))))
            (TypeVarApp
              (TV (Ident {unpackIdent = "m"}))
              [TypeVar (TV (Ident {unpackIdent = "b"}))]
            )
          )
          (TypeVarApp
            (TV (Ident {unpackIdent = "m"}))
```

```
            [TypeVar (TV (Ident {unpackIdent = "b"}))]
          )
        )}
```

`TypVarApp` wurde hierbei nachträglich in free-theorems eingebaut, da es ursprünglich nicht vorgesehen war, Typkonstruktorvariablen zu benutzen.