113: newtype の実装

↑ up

- issued: 2020-10-18
- 分類： B 機能追加
- status: Open

概要

newtype を実装する。

なんか難しそうな気がしていたが、型検査においてはコンストラクタが 1 つだけ引数をとるような抽象データ型と同様に扱いつつ、Core に変換するときには中身そのものと同じように扱えばいいのではないか…と考えたら、できそうな気がしてきた。

…とおもったけど、Core も型付きなので、Core ではまだ newtype はほどけないな。

STG にするときに、ほどくのかな。または、型推論後、newtype の型そのものをほどいてしまえばいいのかもしれない（actual_type のような感じで）

このあたりを考えて、実装しよう。

調査ログ

2020-10-27 (Tue)

「すごい H〜」より、newtype の使用例に関する testcases を用意

- hellome1.hs：data の場合は helloMe undefined がアボートするす
- hellome2.hs：helloMe _ = "hello" の場合は "hello" になる

以上は、現状でも期待通りに動作する。

- hellome3.hs：newtype の場合

これには未対応。

それとは別に、newtype Pair b a = Pair (a, b) に関するものも準備。newtype でなく、data で記述したものもあわせて都合ふたつ。(newtypepair, newtypepair0)

現状では、data で記述したほう (newtypepair0) もエラーするので、こちらを先に調査すべきかもしれない。

2020-11-10 (Tue)

newtypepair0 を調べてみる。現状では、以下のエラー：

```
src/Rename.hs:(204,9)-(216,48): Non-exhaustive patterns in function renTy
```

ソースを修正して、このエラーになったときの引数を表示するするようにした：

```
Non-exhaustive patterns in renTy: TupleTy [Tyvar (Name {origName = "a", namePos = (1,23), isConName
```

以下二か所を修正：

```
diff --git a/compiler/src/Rename.hs b/compiler/src/Rename.hs
index f10eb1f..752bdc5 100644
--- a/compiler/src/Rename.hs
+++ b/compiler/src/Rename.hs
@@ -215,6 +215,15 @@ scanDecls ds = do
         renTy (A.ListTy t) = do rt <- renTy t
                                 return $ list rt

+        renTy (A.TupleTy ts) = do
+          let tcn = "Prelude.(" ++ take (length ts - 1) (repeat ',') ++ ")"
+              tk = foldr Kfun Star (replicate (length ts) Star)
+              tc = TCon (Tycon tcn tk)
+          ts' <- mapM renTy ts
+          return $ foldl' TAp tc ts'
+
+        renTy t = error $ "Non-exhaustive patterns in renTy: " ++ show t
+
         parseConsts (n, ts) = do
           qn <- renameVar n
           let aty = length ts
@@ -645,6 +654,8 @@ kiExpr t@(A.AppTy _ _) dict = dict ++ kiexpr' t []
         kiexpr' (A.Tycon _) ds = ds
         kiexpr' (A.Tyvar n) ds =
           (origName n, foldr Kfun Star (replicate (length ds) Star)):ds
+        kiexpr' (A.ParTy t) ds = kiexpr' t ds
+        kiexpr' t ds = error $ "kiexpr' :" ++ show t ++ " " ++ show ds

 kiExpr (A.Tyvar n) dict     = dict ++ [(origName n, Star)]
```

これでも、newtypepair0.hs では、deriving Show の処理がうまくいかなくて NG。Show インスタンスを明示的に定義するように変更した newtypepair0b.hs では、Functor インスタンスの定義がうまく処理されていないようで、これまた NG.

さらに簡略化した newtypepair0c.hs は通った。

2020-12-29 (Tue)

newtypepair0b.hs をコンパイルしたときの様子：

```
$ bunny testrun newtypepair0b.hs
/home/unno/bunny/0.9.0/bin/bunnyc -d ./jout/newtypepair0b --xno-implicit-prelude /home/unno/bunny/0.
/home/unno/bunny/0.9.0/bin/bunnyc -d ./jout/newtypepair0b --xlibrary-path /home/unno/bunny/0.9.0/lib
newtypepair0b/Main.java:71: error: cannot find symbol
      Expr t14 = (Expr) new AtomExpr(new Dict(new Dict_36_Prelude_46_Integer_64_Prelude_46_Functor()
                                                          ^
  symbol:   class Dict_36_Prelude_46_Integer_64_Prelude_46_Functor
  location: class Main
1 error
testrun: failed to compile the java files.
```

--ddump-core の一部：

```
---- ddumpCore ----
(Main.main :: (Prelude.IO Prelude.())) =
    (((Prelude.print :: ([Prelude.Show t3] :=> (t3 -> (Prelude.IO Prelude.()))))
        (CompositDict ${Main.Pair Prelude.Show} [${Prelude.Integer Prelude.Show},${Prelude.Integer P
      ((((Prelude.fmap :: ([Prelude.Functor TVar (Tyvar "t4" (Kfun Star Star))] :=> ((t5 -> t6) -> (
           (CompositDict ${Main.Pair Prelude.Functor} [${Prelude.Integer Prelude.Functor}]))
         let
             (Main.l6.l0.F :: ([Prelude.Num t11] :=> (t11 -> t11))) =
                 \(Main.l6.l0.F.DARG0 :: ^^c3^^84) ->
                     \(_Main.l6.l0.F.U1 :: ([Prelude.Num t7] :=> t7)) ->
                         ((((Prelude.* :: ([Prelude.Num t9] :=> (t9 -> (t9 -> t9))))
                               (Main.l6.l0.F.DARG0 :: ^^c3^^85))
                           (_Main.l6.l0.F.U1 :: ([Prelude.Num t7] :=> t7)))
                          (((Prelude.fromInteger :: ([Prelude.Num t10] :=> (Prelude.Integer -> t10
                               (Main.l6.l0.F.DARG0 :: ^^c3^^85))
                           (100 :: Prelude.Integer)))
```

```
  in
    ((Main.l6.l0.F :: ([Prelude.Num t12] :=> (t12 -> t12)))
      ${Prelude.Integer Prelude.Num}))
((Main.Pair :: (((Prelude.(,) t13) t14) -> ((Main.Pair t14) t13)))
  (((Prelude.(,) :: (t15 -> (t16 -> ((Prelude.(,) t15) t16))))
      (((Prelude.fromInteger :: ([Prelude.Num t17] :=> (Prelude.Integer -> t17)))
          ${Prelude.Integer Prelude.Num})
        (2 :: Prelude.Integer)))
    (((Prelude.fromInteger :: ([Prelude.Num t18] :=> (Prelude.Integer -> t18)))
        ${Prelude.Integer Prelude.Num})
      (3 :: Prelude.Integer))))))
```

fmap に渡されている複合辞書 `CompositDict ${Main.Pair Prelude.Functor [$Prelude.Integer Prelude.Functor]` が怪しいですね。`instance Functor (Pair c)` は c が Functor であることを要請しない。このあたりの処理に、なんか決め打ちの変な実装がありそう。