068: pairbind.hs で qname not found

↑ up

- issued: 2020-05-09
- 分類： A サンプルコードが fail
- status: Closed (2020-05-14)

概要

divMod のデフォルト実装がエラーしたことで発覚。

```
where (p, q) = (1, 2)
```

のような束縛がうまくいかない。

```
where (,) p q = (1, 2)
```

でもだめ。それぞれにテストケースを用意した (pairbind2, pairbind)

調査ログ

2020-05-09 (Sat)

左辺がコンストラクタのパターンの場合（これに関する記述は、Haskell 2010 Language Report のどこにあるんだろ[*1]）には、複数の binding に変換してやればいいように思われる：

```
C a b c ... = rhs
```

⇒

```
a = let e = rhs
      in case e of
        C x y z ... -> x

b = let e = rhs
      in case e of
```

[*1] thih の 11.6.3 にはこの変換に関する記載あり

```
        C x y z ... -> y

c = let e = rhs
      in case e of
        C x y z ... -> z
```

(この方針で patbind.hs を手で変換したのが patbind2.hs)

現状の実装では、renDecl や、そこから呼ばれている renFExp が、関数名が isConName である場合を考慮していない。

およそ、以下のようなことをやっていくべき。

- 現状の renDecl, renFExp の動作を整理する
- InfixExp は fixity resolution をすませた上で、FunAppExp に変換して、統一的に処理するようにしたい
- その後、本件のケース（関数名が isConName だった場合）に対処する

また、関連 issue を 2 つたてた: (069, 070)

2020-05-10 (Sun)

この変換は as-pattern にいったん直してから、as-pattern として扱った方がいいような気がしてきた。

つまり、`C a b c ...= rhs` は、いったん `e@(C a b c ...) = rhs` に変換してから、以下の変換を施す:

```
e@(C a b c ...) = rhs

⇒
e = rhs

a = case e of
        C x y z ... -> x

b = case e of
        C x y z ... -> y

c = case e of
        C x y z ... -> z
```

2020-05-11 (Mon)

まずは As-pattern に対処してから、上述のように、パターンを as-pattern に変換する処理を追加したい。なので、まずは、明にかかれた As-pattern のケースを 2 つ追加した。

- pairbind3.hs
- pairbind4.hs

3, 4 の順に対処していけると思う。

scanValueDecl2 で、AsPat に対する限定的な対処をいれて、pairbind3, pairbind4 のケースに対しては As pattern から上述のような変換をするようにした。

ところが、その後でエラー。もともとは、TupleExp も FunAppExp に変換して統一的に扱うつもりだったが、調査のため、現状はわけて処理している。

pairbind3.hs の方は、`findCMs failed: "Prelude.(,)"`, pairbind4.hs の方は、`dictionary not found: ("Prelude.Sho` となる。

pairbind4.hs と同様の変換をあらかじめ手でかいておいた、pairbind4b.hs でも同じく `dictionary not found: ("Prelude` となるため、今回追加した変換に問題があるのではなく、辞書渡し変換の潜在的な障害だと思われる。

なお、pairbind4b.hs で、x に型注釈をつける (pairbind4b.hs) と通る。

2020-05-14 (Thu)

pairbind4.hs に関して、この現象が再現する最小コードがどうなるか考えてみた。以下の形にまで変形しても、エラーは再現した：

```
main = do print p
          where
            x = (1, 2)
            p = case x of
              (a, b) -> a
```

つまり、Rename に追加した変換に問題があるのではなく、期待通り変換しても型変換に失敗しているということ。

なお、case 式に x ではなく、(1, 2) を直接与えるようにしたら通る：

```
main = do print p
          where
            p = case (1, 2) of
              (a, b) -> a
```

それぞれの core0 を示す。まず、うまくいかない方（case に x を与えた方）：

```
---- ddumpCore ----
(Main.main :: (Prelude.IO ())) =
    let
        (Main.l1.l0.p :: v1873) =
            let
                (Main.l1.l0.l1.l0.F :: (((Prelude.(,) t6) t7) -> t6)) =
                    \(_Main.l1.l0.l1.l0.F.U1 :: ((Prelude.(,) t0) t1)) ->
                        case (_Main.l1.l0.l1.l0.F.U1 :: ((Prelude.(,) t0) t1)) (_Main.l1.l0.l1.l0.F.
                            Prelude.(,) (_Main.l1.l0.l1.l0.F.U2 :: t2) (_Main.l1.l0.l1.l0.F.U3 :: t3
                                (_Main.l1.l0.l1.l0.F.U2 :: t2)



            in
              ((Main.l1.l0.l1.l0.F :: (((Prelude.(,) t8) t9) -> t8))
                (Main.l1.l0.x :: ([Prelude.Num v1860,Prelude.Num v1861] :=> ((Prelude.(,) v1860) v18

        (Main.l1.l0.x :: ([Prelude.Num v1860,Prelude.Num v1861] :=> ((Prelude.(,) v1860) v1861))) =
            (((Prelude.(,) :: (t14 -> (t15 -> ((Prelude.(,) t14) t15))))
                ((Prelude.fromInteger :: ([Prelude.Num t16] :=> (Prelude.Integer -> t16)))
                  (1 :: Prelude.Integer)))
              ((Prelude.fromInteger :: ([Prelude.Num t17] :=> (Prelude.Integer -> t17)))
                (2 :: Prelude.Integer)))



    in
      ((Prelude.print :: ([Prelude.Show t20] :=> (t20 -> (Prelude.IO ()))))
        (Main.l1.l0.p :: v1873))
```

つぎに、うまくいった方（case に (1, 2) を直接書いた方）:

```
---- ddumpCore ----
(Main.main :: (Prelude.IO ())) =
    let
        (Main.l1.l0.p :: ([Prelude.Num v1866] :=> v1866)) =
            let
                (Main.l1.l0.l0.l0.F :: (((Prelude.(,) t7) t8) -> t7)) =
                    \(_Main.l1.l0.l0.l0.F.U1 :: ((Prelude.(,) t1) t2)) ->
```

```
                    case (_Main.l1.l0.l0.l0.F.U1 :: ((Prelude.(,) t1) t2)) (_Main.l1.l0.l0.l0.F.
                        Prelude.(,) (_Main.l1.l0.l0.l0.F.U2 :: t3) (_Main.l1.l0.l0.l0.F.U3 :: t4
                            (_Main.l1.l0.l0.l0.F.U2 :: t3)


          in
            ((Main.l1.l0.l0.l0.F :: (((Prelude.(,) t9) t10) -> t9))
              (((Prelude.(,) :: (t11 -> (t12 -> ((Prelude.(,) t11) t12))))
                  ((Prelude.fromInteger :: ([Prelude.Num t13] :=> (Prelude.Integer -> t13)))
                    (1 :: Prelude.Integer)))
                ((Prelude.fromInteger :: ([Prelude.Num t14] :=> (Prelude.Integer -> t14)))
                  (2 :: Prelude.Integer))))


  in
    ((Prelude.print :: ([Prelude.Show t16] :=> (t16 -> (Prelude.IO ()))))
      (Main.l1.l0.p :: ([Prelude.Num v1866] :=> v1866)))
```

Main.l1.l0.p の型がちがうなぁ。これは、Assump (Typing の出力) 時点ですでに異なっているのだが、Thih 自体が間違っているとは思えない（これまでそんなことは一度もなかった)。そういえば、case 式を Typing に与えるとき、自前でなんかしてたかなぁ…。

そのあたりが怪しい？

問題回避

ひとまず、case 式に直接右辺の式をあたえる (`p = case (1, 2) of ...` にする) ことで問題を回避し、as-pattern の実装はすすめることにしたい。根本解決になっていないし、右辺が複雑な場合（ガードがあったり、where 節があったり）には、対応できないのだが。

この問題は切り出して、別 issue とする。

調査継続

pairbind3.hs でエラーする (findCMs failed) 原因は、Predefined における漏れだった。以下のように primConsMem の宣言に pairCfun を追加することで解決：

```
--- a/compiler/src/PreDefined.hs
+++ b/compiler/src/PreDefined.hs
@@ -199,7 +199,7 @@ overloadedCfun = "#overloaded#" :>:
   Forall [Star, Star] ([] :=> (TGen 0 `fn` tString `fn` TGen 1))
```

```
 primConsMems :: [Assump]
-primConsMems  = [ unitCfun, nilCfun, consCfun
+primConsMems  = [ unitCfun, nilCfun, consCfun, pairCfun
```

解決

上記の対処にくわえて、コンストラクタ適用式を As-pattern に変換する処理を加えることで、本 issue で想定していた testcases はクリア。

- pairbind : sample199
- pairbind2: sample200
- pairbind3: sample201
- pairbind4: sample202
- lsbind: sample203
- patbind2: samle204

本件ワークアラウンドへの正式対処は、別 issue 073, 074 で継続することとして、本件はクローズ。