083: litpattern.hs で renPat: LitExp (LitInteger 0 (1,3))

↑ up

- issued: 2020-05-18
- 分類： A サンプルコードが fail
- status: Closed (2020-05-20)

概要

lib/Prelude.hs に gcd を定義しようとして発覚。パターンに整数リテラルを書くとエラー。
litpattern.hs

```
f 0 = "zero"
f _ = "non-zero"
main = do putStrLn $ f 1
          putStrLn $ f 0
```

類似： litpattern2.hs, litpattern3.hs
なお、似ているが case 式の場合には異る箇所でエラーしたため、別 issue (084) とした。

概要

2020-05-18 (Mon)

最初、リテラルを疑似コンストラクタで表現するのかなとも思ったが、Bounded とはいえ値の個数がおおく（それはなんとかしたとしても）、数値リテラルは多相なので、うまくいきそうにない。
そのことを考慮すると、== 演算子をもちいるような形に desuger してやるしかないのではないか。
参考として、GHC がどうしているのかを、`ghc -ddump-ds-preopt testcases/litpattern.hs` で確認：

```
f :: forall a. (Eq a, Num a) => a -> [Char]
[LclId]
f = \ (@ a_a211)
      ($dEq_a2hL :: Eq a_a211)
      ($dNum_a2hM :: Num a_a211) ->
      let {
        $dNum_a2ds :: Num a_a211
        [LclId]
        $dNum_a2ds = $dNum_a2hM } in
```

```
      let {
        $dEq_a212 :: Eq a_a211
        [LclId]
        $dEq_a212 = $dEq_a2hL } in
      letrec {
        f_a2du :: a_a211 -> [Char]
        [LclId]
        f_a2du
          = \ (ds_d2kK :: a_a211) ->
              let {
                fail_d2l2 :: GHC.Prim.Void# -> [Char]
                [LclId]
                fail_d2l2
                  = \ (ds_d2l3 [OS=OneShot] :: GHC.Prim.Void#) ->
                      GHC.CString.unpackCString# "non-zero"# } in
              case ==
                     @ a_a211 $dEq_a212 ds_d2kK (fromInteger @ a_a211 $dNum_a2ds 0)
              of wild_00 {
                False -> fail_d2l2 GHC.Prim.void#;
                True -> GHC.CString.unpackCString# "zero"#
              }; } in
      f_a2du
```

単純化すると、次のように書き換えているようだ (litpattern-ds.hs) :

```
f = \x ->
  let
    fail' = "non-zero"
  in
    case (==) x 0 of
      False -> fail'
      True  -> "zero"
```

fail の内容が変数に束縛されていて、複数の個所から呼べるように（上の例では一か所だが）なっているのが、変換のヒントのように見える。

どういうアルゴリズムでこのように変換するかは、考えどころ。

なんとなくだが、以下のように変換してから 064 の問題と統一的に扱うのが良い気がする。