067: showlist2.hs がランタイムに abend

↑ up

- issued: 2020-05-08
- 分類： A サンプルコードが fail
- status: Closed (2020-05-13)

概要

055 の調査のために、再現用の小さなテストプロを作成。
showlist2.hs:

```
$ cat testcases/showlist2.hs
main = do
  putStrLn $ showList [1] ""
    where
      showList xs = (++) (showlist' xs)
        where showlist' [] = "[]"
              showlist' [x] =  "[" ++ show x ++ "]"
              showlist' (x:xs) = "[" ++ foldl (\s t -> s ++ "," ++ show t) (show x) xs ++ "]"
```

これをコンパイル、実行するとランタイムでエラーする。なお、これに似た showlist.hs は大丈夫。
showlist.hs:

```
main = do
  putStrLn $ showList [1] ""
    where showlist' [] = "[]"
          showlist' [x] =  "[" ++ show x ++ "]"
          showlist' (x:xs) = "[" ++ foldl (\s t -> s ++ "," ++ show t) (show x) xs ++ "]"

          showList xs = (++) (showlist' xs)
```

また、showlist2.hs の showlist' に型注釈をつけたもの（showlist3.hs）も Ok.
エラーするケースでは、showlist' 内部における辞書の解決がおかしいようにみえる。
let 式がネストされているケースの多相の解決（DictPass による辞書渡し形式への変換）に難があるようだ。flesh な type value にすべきところをしていないとか、かも。

うまくいくケースとそうでない場合の core0, core を見比べてみよう。

調査ログ

2020-05-12 (Tue)

だめな場合の core を確認すると、DARG0 を渡すべきところで、$Prelude.Char Prelude.Show を渡しているよう。型注釈をつけた showlist3.hs では、$Prelude.Integer Prelude.Show がその場で生成されているので OK。

ただ、まだ例が大きすぎてみるのが大変だし、被疑個所が特定しづらい。そこで、もっと小さくしていって、エラーが出る場合と出ない場合の境界をあきらかにしたい。

どうやら、これが最小再現ケースではないかな (t067.hs) :

```
main = do
  putStrLn $ f 1 ""
    where
      f xs str = str ++ (show' xs)
        where show' x =  show x
```

show' はここまで単純でも再現する。でも、let が nest されていて、show' の型が決まらない（多相になる）のは必須だと思われる。

instancee (Show a) => Show [a] の実装において発覚した問題だが、リストは関係なかったよう。また、元の例ではパターンマッチコンパイル結果の case 式が頻出していて、その複雑な場合に発生する問題かなと恐れたのですが、そうでもなかった。

なお、`f` はこれ以上単純化する、たとえば、`f xs str = show' xs` のようにすると、もう、再現しない。

t067.hs の問題の core は次の通り :

```
---- ddumpCore ----
(Main.main :: (Prelude.IO ())) =
    let
        (Main.l1.l0.f :: ([Prelude.Show t5] :=> (t5 -> ([Prelude.Char] -> [Prelude.Char])))) =
            \(Main.l1.l0.f.DARG0 :: ^^c3^^84) ->
                \(_Main.l1.l0.f.U1 :: ([Prelude.Show t0] :=> t0)) (_Main.l1.l0.f.U2 :: ([Prelude.Sho
                    let
                        (Main.l1.l0.l0.l0.show' :: ([Prelude.Show t3] :=> (t3 -> [Prelude.Char]))) =
                            \(Main.l1.l0.l0.l0.show'.DARG0 :: ^^c3^^84) ->
                                \(_Main.l1.l0.l0.l0.show'.U1 :: ([Prelude.Show t1] :=> t1)) ->
                                    (((Prelude.show :: ([Prelude.Show t2] :=> (t2 -> [Prelude.Char])
                                        ${Prelude.Char Prelude.Show})
```