065: listofstr.sh などで Non-exhaustive patterns in function simpleTy2dict

↑ up

- issued: 2020-05-07
- 分類： A サンプルコードが fail
- status: Closed (2020-05-13)

概要

CompositDict の中身が再び CompositDict になるようなケースに対応したコードになっていないので、リストのリストのような場合に Show がうまくいかないだろうなと予想していた。

そういった事象を確認するためのテストケースを追加：

- listofstr.hs
- listoflist.hs
- complexpair.hs
- listofpairs.hs
- nestedlist.hs

いずれも、同じエラーとなる：

```
$ ./tcheck testcases/listoflist.hs
# 1. test-compile
source file: testcases/listoflist.hs
dst dir: /listoflist
doCompile ... done.
implicitPrelude ... done.
doCompile ... bunnyc: src/DictPass.hs:(246,16)-(256,72): Non-exhaustive patterns in function simpleT
```

調査ログ

2020-05-07 (Thu)

辞書への変換が appTy2dict と simpleTy2dict に分かれていたのを統合（同じ名前にしただけ）、再帰的に用いるようにした。

その結果、listofstr, listofpairs は OK だったが、ほかの 3 つはランタイムに abend.

listofstr が ok で他がダメなもののうち、toJString で assertion error しているものは、Char オブジェクトが未評価であるのに耐えられていないんではないかな？

継続調査。

2020-05-08 (Fri)

listoflist, nestedlist でエラーしているのは、RTLib.toJString におけるアサーションだった。実際、どんなオブジェクトが来ているのか表示するようにしたところ、BoxedIntegerObj が来ていたようだ。つまり、昨夜の予想は外れ。(以下の例は listoflist を実行したあとの trace.txt):

```
$ cat jout/trace.txt
Not a BuxedCharObj:AtomExpr(a=Var(obj=2 :: Integer))
Exception in thread "main" java.lang.AssertionError
        at jp.ne.sakura.uhideyuki.brt.runtime.RTLib.toJString(RTLib.java:88)
        at jp.ne.sakura.uhideyuki.brt.runtime.PutStrLnFunc.call(RTLib.java:19)
        at jp.ne.sakura.uhideyuki.brt.runtime.EvalApply.evalFun(RT.java:239)
        at jp.ne.sakura.uhideyuki.brt.runtime.EvalApply.runStep(RT.java:60)
        at jp.ne.sakura.uhideyuki.brt.runtime.EvalApply.eval(RT.java:18)
        at jp.ne.sakura.uhideyuki.brt.runtime.RT.eval(RT.java:288)
        at Sample.main(Sample.java:6)
```

また、listofpairs の場合は以下のようになっており、

```
$ cat jout/trace.txt
Exception in thread "main" java.lang.AssertionError
        at jp.ne.sakura.uhideyuki.brt.runtime.EvalApply.CaseAny(RT.java:182)
        at jp.ne.sakura.uhideyuki.brt.runtime.EvalApply.evalCase(RT.java:121)
        at jp.ne.sakura.uhideyuki.brt.runtime.EvalApply.runStep(RT.java:45)
        at jp.ne.sakura.uhideyuki.brt.runtime.EvalApply.eval(RT.java:18)
        at jp.ne.sakura.uhideyuki.brt.runtime.RT.eval(RT.java:288)
        at jp.ne.sakura.uhideyuki.brt.runtime.RTLib.tail(RTLib.java:74)
        at jp.ne.sakura.uhideyuki.brt.runtime.RTLib.toJString(RTLib.java:92)
        at jp.ne.sakura.uhideyuki.brt.runtime.PutStrLnFunc.call(RTLib.java:19)
        at jp.ne.sakura.uhideyuki.brt.runtime.EvalApply.evalFun(RT.java:239)
        at jp.ne.sakura.uhideyuki.brt.runtime.EvalApply.runStep(RT.java:60)
        at jp.ne.sakura.uhideyuki.brt.runtime.EvalApply.eval(RT.java:18)
        at jp.ne.sakura.uhideyuki.brt.runtime.RT.eval(RT.java:288)
        at Sample.main(Sample.java:6)
```

該当箇所は、CaseAny で scrut が変数だったときに、defaultAlt が null でないことを検査している個所：

```
    private Boolean CaseAny(){
        CaseExpr e = (CaseExpr) code;
        Expr scrut = e.scrut;
        Alt[] alts = e.alts;

        if (scrut.isValue()){
            DefaultAlt dalt = getDefaultAlt(alts);
            assert dalt != null;   // 182 行目のアサーションはココ！
            AtomExpr[] a = new AtomExpr[1];
            a[0] = (AtomExpr) scrut; // it is safe when scrut.isLitOrValue()
            code = new FunAppExpr(dalt.e, a, -1);
            return true;
        }
        return false;
    }
```

いずれも、これらだけでは原因はわからず。もう少し小さい再現例で調査したい。

⇒ listoflist2, listoflist3 を追加。`[[3]]` は Ok で `[[3], [4]]` で NG。(listofpairs もリストの要素をひとつにした listofpair1 なら ok)

意外に根深い（もしかしたら、今の設計ではダメとか）問題かもしれないなぁ。

とりあえず Core を見てみよう。

listoflist2 の Core:

```
(Main.main :: (Prelude.IO ())) =
    (((Prelude.print :: ([Prelude.Show t0] :=> (t0 -> (Prelude.IO ()))))
        (CompositDict ${Prelude.[] Prelude.Show} [(CompositDict ${Prelude.[] Prelude.Show} [${Prelud
      (((Prelude.: :: (t1 -> ([t1] -> [t1])))
          (((Prelude.: :: (t2 -> ([t2] -> [t2])))
              (((Prelude.fromInteger :: ([Prelude.Num t3] :=> (Prelude.Integer -> t3)))
                  ${Prelude.Integer Prelude.Num})
                (3 :: Prelude.Integer)))
            (Prelude.[] :: [t4])))
        (Prelude.[] :: [t5])))
```

listoflist3 の Core:

```
(Main.main :: (Prelude.IO ())) =
    (((Prelude.print :: ([Prelude.Show t0] :=> (t0 -> (Prelude.IO ()))))
```

```
  (CompositDict ${Prelude.[] Prelude.Show} [(CompositDict ${Prelude.[] Prelude.Show} [${Prelud
(((Prelude.: :: (t1 -> ([t1] -> [t1])))
   (((Prelude.: :: (t2 -> ([t2] -> [t2])))
      (((Prelude.fromInteger :: ([Prelude.Num t3] :=> (Prelude.Integer -> t3)))
         ${Prelude.Integer Prelude.Num})
       (3 :: Prelude.Integer)))
     (Prelude.[] :: [t4])))
  (((Prelude.: :: (t5 -> ([t5] -> [t5])))
     (((Prelude.: :: (t6 -> ([t6] -> [t6])))
        (((Prelude.fromInteger :: ([Prelude.Num t7] :=> (Prelude.Integer -> t7)))
           ${Prelude.Integer Prelude.Num})
         (4 :: Prelude.Integer)))
       (Prelude.[] :: [t8])))
    (Prelude.[] :: [t9])))))
```

Pair の方は、ネストを深くしても大丈夫っぽい（nestedpairs, nestedpairs2)。List の Show に特有の問題か (foldl とかでがつんとやってるのが良くない？？ んなことないか？)。

ひとまず、通ったものは test/samples に追加しておこう：

- listofstr: sample180
- complexpair: sample181
- nestedpairs: sample182
- nestedpairs2: sample183

055 を先に片づけたほうがいいのかもしれない。

被疑個所？：λ抽象への辞書渡し？

あからさまに怪しかったのは、以下の Prim.integerShow のところ。っていうか、これが呼ばれていること自体がおかしいのだが…。

```
instance Show Integer where
  show = Prim.integerShow
  showList xs = (++) (showlist' xs)
    where showlist' [] = "[]"
          showlist' [x] =  "[" ++ Prim.integerShow x ++ "]"
          showlist' (x:xs) = "[" ++ foldl (\s t -> s ++ "," ++ Prim.integerShow t) (show x) xs ++ "]
```

この Prim.integerShow を show に変えても、症状はかわらない。そういえば、λ項に辞書を渡していただろうか？

λ項がくさいと思ったのは、勘違いだったかもしれない。
やっぱり、まず 055 に取り組んだ方がよさそう。

解決

067 対処によって、本件も解決。

- listoflist.hs : sample196
- listofpairs.hs: sample197
- nestedlist.hs: sample198
- listoflist2, listoflist3, listofpair1: 調査要だったので、廃止