045: DictPass における defaulting がその場しのぎ

↑ up

- issued: 2020-04-22
- 分類： 分類：C 改善項目
- status: Open

概要

DictPass における defaulting がその場しのぎの実装になっている。

調査ログ

2020-04-22 (Wed)

044 対処のなかで、DictPass 中で defaulting 相当のことを実施する必要が生じて、その場しのぎの実装を行った。Type Check を通してはいけないケースを通してしまうようなダメさはないのだが、実装としてもいいかげんだし、特殊なケースにしか対応していないので、あとで改善が必要。

以下に、おこなった内容を記録しておく。

まず、044 の対処として、TrCore.trExpr2 が整数リテラルにつける型を、Integer から Num a => a に変更

そうすることで、DictPass における findApplyDict が失敗するようになった。この変更前は、Integer を探し当てていたケースが、総称型にいきあたってしまうため。

そこで、DictPass の TcState に、このような Num a => a になっている整数リテラルにであったら、それを記録するリストを追加：

```
-data TcState = TcState { tcCe    :: ClassEnv
-                       , tcPss   :: [(Pred, Id)]
-                       , tcSubst :: Subst
-                       , tcNum   :: Int
+data TcState = TcState { tcCe           :: ClassEnv
+                       , tcPss          :: [(Pred, Id)]
+                       , tcSubst        :: Subst
+                       , tcNum          :: Int
+                       , tcIntegerTVars :: [Type]
                        }
```

つぎに、DictPass.getTy にて、整数リテラルに出会った際に、この追加したリストに記録していく。

```
+getTy e@(Lit (LitInt  _ qt@(_ :=> v))) = do
+  st <- get
+  let tvars = tcIntegerTVars st
+      st' = st{tcIntegerTVars = (v:tvars)}
+  put st'
+  return qt
```

最後に、findApplyDict において、このリストに記録されている型変数と同一化されるような型変数については、Integer 型の辞書を生成するようにした。

```
          findApplyDict e (qv :=> t') (_ :=> t) = do
            unify' t' t
            s <- getSubst
+           itvars <- tcIntegerTVars <$> get
+           let itvars' = fmap (apply s) itvars
            let  mkdicts [] ds = return ds
                 mkdicts (IsIn n2 (TVar x) : qs) ds =
                   case apply s (TVar x) of
                     (TCon (Tycon n1 _)) -> mkdicts qs (Var (DictVar n1 n2) : ds)
-                    y -> do v <- lookupDictArg (n2, x)
-                            case v of
-                              Nothing -> error ("Error: dictionary not found: "
-                                                ++ n ++ ", " ++ show (x,n2,y))
-                              Just v' -> mkdicts qs (Var v' : ds)
+                    y | y `elem` itvars'
+                        -> mkdicts qs (Var (DictVar "Prelude.Integer" n2) : ds)
+                      | otherwise
+                        -> do v <- lookupDictArg (n2, x)
+                              case v of
+                                Nothing -> error ("Error: dictionary not found: "
+                                                  ++ n ++ ", " ++ show (x,n2,y,itvars))
+                                Just v' -> mkdicts qs (Var v' : ds)
                 mkdicts _ _ = error "mkdicts: must not occur"
            dicts <- mkdicts qv []
            return (foldl App e dicts)
```

これで、整数リテラルが曖昧なケースに "Integer" に defaulting するケースに限っては、一応正しく動くのではないかと思われる。

2020-04-28 (Tue)

うえで「一応正しく動くのではないか」と書いたのは、誤り。一見うごいているように見えるが、このやり方では正しくない (実装としてその場しのぎであるだけでなく、間違っている)。

それが明らかな例が g2.hs:

```
$ cat testcases/g2.hs
g x = x + 1

a :: Int
a = 9

main = putStrLn $ show $ g a
```

これの core は次のようになっている :

```
---- ddumpCore ----
(Main.main :: (Prelude.IO ())) =
    ((Prim.putStrLn :: ([Prelude.Char] -> (Prelude.IO ()))) $
      (((Prelude.show :: ([Prelude.Show t4] :=> (t4 -> [Prelude.Char])))
          ${Prelude.Int Prelude.Show}) $
        (((Main.g :: ([Prelude.Num t5] :=> (t5 -> t5)))
            ${Prelude.Int Prelude.Num})
          (Main.a :: Prelude.Int))))

(Main.g :: ([Prelude.Num t9] :=> (t9 -> t9))) =
    \(Main.g.DARG0 :: ^^c3^^84) ->
        \(_Main.g.U1 :: ([Prelude.Num t6] :=> t6)) ->
            ((((Prelude.+ :: ([Prelude.Num t7] :=> (t7 -> (t7 -> t7))))
                  ${Prelude.Integer Prelude.Num})
                (_Main.g.U1 :: ([Prelude.Num t6] :=> t6)))
              (1 :: ([Prelude.Num t8] :=> t8)))

(Main.a :: Prelude.Int) =
    (9 :: ([Prelude.Num t9] :=> t9))
```

Main.g の型は `Num a => a -> a` なので、この関数は Num クラスの辞書をうけとる。また、呼び出し側（Main.main）では、 a が Int なので、Int の辞書を渡している。そこまではあっている。

だが、現状では Main.g において、 x + 1 の 1 に誤った defaulting を施してしまい、うけっとった辞書ではなく Integer の辞書を (+) にわたしてしまっている。これでも動いているように見えるのは、現在のランタイムにおいては、Int と Integer が実質的に同じであるせい。

defaulting の規則がきめうち (Num a => a を Integer にする）なのはいいとしても、現状の制約されていない整数リテラルにであったら Integer にしてしまうのはまずいので、これを正す必要がある。

これを直さないと、Integer をきちんと多倍長にしたり、Double などを実装したときに破綻する。また、現状 fail しているほかの件にも影響しているかもしれない。(showsPrec がダメな件とかあやしい)

2020-05-01 (Fri)

以下で、lookupDictArg より先に（なんちゃって）defaulting をしているのがいけないように思われる (src/DictPass.hs l.242-) :

```
simpleTy2dict n2 (TVar y)
  | (TVar y) `elem` itvars' =  return (Var (DictVar "Prelude.Integer" n2))
  | otherwise =
    do v <- lookupDictArg (n2, y)
       case v of
         Nothing -> error ("Error: dictionary not found: "
                           ++ n ++ ", " ++ show (n2,y,itvars))
         Just v' -> return (Var v')
```

これを、以下のように変更 :

```
simpleTy2dict n2 (TVar y) =
  do v <- lookupDictArg (n2, y)
     case v of
       Just v' -> return (Var v')
       Nothing | (TVar y) `elem` itvars' ->
                 return (Var (DictVar "Prelude.Integer" n2))
               | otherwise ->
                 error ("Error: dictionary not found: "
                        ++ n ++ ", " ++ show (n2,y,itvars))
```

この変更で、少なくとも g2.hs のケースは正しくなった :

```
(Main.main :: (Prelude.IO ())) =
    ((Prim.putStrLn :: ([Prelude.Char] -> (Prelude.IO ()))) $
```

```
      (((Prelude.show :: ([Prelude.Show t4] :=> (t4 -> [Prelude.Char])))
          ${Prelude.Int Prelude.Show}) $
        (((Main.g :: ([Prelude.Num t5] :=> (t5 -> t5)))
            ${Prelude.Int Prelude.Num})
          (Main.a :: Prelude.Int))))

(Main.g :: ([Prelude.Num t9] :=> (t9 -> t9))) =
    \(Main.g.DARG0 :: ^^c3^^84) ->
        \(_Main.g.U1 :: ([Prelude.Num t6] :=> t6)) ->
            ((((Prelude.+ :: ([Prelude.Num t7] :=> (t7 -> (t7 -> t7))))
                  (Main.g.DARG0 :: ^^c3^^85))
                (_Main.g.U1 :: ([Prelude.Num t6] :=> t6)))
              (1 :: ([Prelude.Num t8] :=> t8)))

(Main.a :: Prelude.Int) =
    (9 :: ([Prelude.Num t9] :=> t9))
```

メモ： とりあえず、これで少しマシになったのだけど、リテラルを特別に扱っている時点でよくない。これでは fromIntegral などに対処できない。型で処理すべき。

2020-05-04 (Mon)

リテラルに由来している `Num [t] => t` にしか対応していないことで strlen.hs が動かなかった 054 ので、現状の実装を流用しつつ、これに対応した。

現状では、TcState に tcIntegerTVars という [Type] 型のフィールドを設け、Num [t] -> t であるところの整数リテラルの型変数をこれに登録していた。

```
getTy e@(Lit (LitInt  _ qt@(_ :=> v))) = do
  st <- get
  let tvars = tcIntegerTVars st
      st' = st{tcIntegerTVars = (v:tvars)}
  put st'
  return qt
```

辞書を探すのに失敗したときには、これを参照して、もし見つかれば（やっつけ）defaulting で Integer 型としている。

```
                simpleTy2dict n2 (TVar y) =
                  do v <- lookupDictArg (n2, y)
```