060: wheresample.hs で IntegerShowFunc: must not occur

↑ up

- issued: 2020-05-05
- 分類： A サンプルコードが fail
- status: Closed (2020-05-06)

概要

wheresample.hs がランタイムで IntegerShowFunc: must not occur となって異常終了する。
wheresample.hs:

```
main = print x
  where x = 13
```

これは、059 の調査で発覚した。関連テストケース：

- arithmeticseq.hs
- aseq.hs

調査ログ

2020-05-05 (Tue)

wheresample.hs の core は次のようになっている：

```
---- ddumpCore ----
(Main.main :: (Prelude.IO ())) =
    let
        (Main.l0.l0.x :: ([Prelude.Num v1178] :=> v1178)) =
            \(Main.l0.l0.x.DARG0 :: ^^c3^^84) ->
                (13 :: ([Prelude.Num t1] :=> t1))


    in
      (((Prelude.print :: ([Prelude.Show t2] :=> (t2 -> (Prelude.IO ()))))
          ${Prelude.Integer Prelude.Show})
        (Main.l0.l0.x :: ([Prelude.Num v1178] :=> v1178)))
```

`x :: Num t => t` が多相なので、これに辞書を渡そうとしているのはむしろ、正しい。整数リテラルをなんちゃってで定数のように扱っているのが間違い。

いま思うと、Core の LitInt の型を Num t => t にしたのは、間違いだった。

`Typing.LitInt i` の型は `Num t => t` でいいのだが、これは、Core に変換するときに、`App fromInteger (LitInt i Integer)` のようにすべきだったと思われる。

つまり、Core.Literal の型は Qual Type に直す必要はなく、Type でよかった。

まず、ランタイムはなるべく現状のまま（実質的に Int と Integer の区別はまだついていない）でも、本件の対策はできるので、そちらをやってしまいたい。

fromInteger を、Integer むけには id, Int むけには Prim.intFromInteger（これは新しく用意）とし、DictPass を正しく直す。

まずは、fromInteger だけ用意してしまおう。

2020-05-06 (Wed)

やるべきことは以下：

- Core.Literal の第二フィールドは、Qual Type ではなく Type にもどす
- Core における LitInt の型は Num t => t ではなく Integer とし、Ty.Literal (LitInt x) は TrCore によって App fromInteger (Lit (LitInt x tInteger)) のように変換
- DictPass において、定数項 (arity = 0) にも辞書を渡すようにする

それぞれ、以下のように修正（他にも PPCore など関連して修正）：

```
diff --git a/compiler/src/Core.hs b/compiler/src/Core.hs
index 2b0b272..d884aba 100644
--- a/compiler/src/Core.hs
+++ b/compiler/src/Core.hs
@@ -12,10 +12,10 @@ data Var = TermVar Id (Qual Type)
          {- | TypeVar Id Kind -} -- unused
          deriving (Show, Eq)

-data Literal = LitInt  Integer (Qual Type)
-             | LitChar Char    (Qual Type)
-             | LitFrac Double  (Qual Type)
-             | LitStr  String  (Qual Type)
+data Literal = LitInt  Integer Type
+             | LitChar Char    Type
+             | LitFrac Double  Type
+             | LitStr  String  Type
              deriving (Show, Eq)
```

```
 data Expr = Var Var

diff --git a/compiler/src/DictPass.hs b/compiler/src/DictPass.hs
index 1b201bf..4a6f226 100644
--- a/compiler/src/DictPass.hs
+++ b/compiler/src/DictPass.hs
@@ -107,16 +107,10 @@ getTy (Var (TermVar _ qt@(ps :=> _))) =
                                          checkPreds ps
                | otherwise = checkPreds ps

-getTy (Lit (LitChar _ qt)) = return qt
-getTy (Lit (LitFrac _ qt)) = return qt
-getTy (Lit (LitStr  _ qt)) = return qt
-
-getTy e@(Lit (LitInt  _ qt@(_ :=> v))) = do
-  st <- get
-  let tvars = tcIntegerTVars st
-      st' = st{tcIntegerTVars = (v:tvars)}
-  put st'
-  return qt
+getTy (Lit (LitChar _ t)) = return ([] :=> t)
+getTy (Lit (LitFrac _ t)) = return ([] :=> t)
+getTy (Lit (LitStr  _ t)) = return ([] :=> t)
+getTy (Lit (LitInt  _ t)) = return ([] :=> t)

 getTy (App f e) = do
   (qf :=> tf) <- getTy f
@@ -158,10 +152,10 @@ tyScrut s as = do

         altty (LitAlt l, _, _) = return $
                                  case l of
-                                   LitInt  _ ( _ :=> t) -> t
-                                   LitChar _ ( _ :=> t) -> t
-                                   LitFrac _ ( _ :=> t) -> t
-                                   LitStr  _ ( _ :=> t) -> t
+                                   LitInt  _ t -> t
+                                   LitChar _ t -> t
+                                   LitFrac _ t -> t
+                                   LitStr  _ t -> t
```

```
         altty _ = do n <- newNum
                      return $ TVar (Tyvar ("a" ++ show n) Star)


 trExpr2 (Ty.Lit (Ty.LitInt n)) = do
   v <- newTVar' Star
-  return (Lit (LitInt n ([IsIn "Prelude.Num" v] :=> v)))
+  let qty = [IsIn "Prelude.Num" v] :=> (tInteger `fn` v)
+      f = Var (TermVar "Prelude.fromInteger" qty)
+      i = Lit (LitInt n tInteger)
+  return (App f i)

 trExpr2 (Ty.Ap e1 e2) = do
   e1' <- trExpr2 e1



@@ -212,7 +206,7 @@ mkTcState ce pss subst num =
 tcExpr :: Expr -> Qual Type -> TC Expr

 tcExpr e@(Var (TermVar n (qv :=> t'))) qt -- why ignore qs?
-  | null qv || isTVar t' e || isArg n {- todo:too suspicious! -} = return e
+  | null qv || {- isTVar t' e || -} isArg n {- todo:too suspicious! -} = return e
   | otherwise = findApplyDict e (qv :=> t') qt
   where isTVar x@(TVar _) y = True
         isTVar x y          = False
diff --git a/compiler/src/PPCore.hs b/compiler/src/PPCore.hs
index d87ac91..2ab1d27 100644
```

最後、なぜ t' が Tyvar だったとき除外していたのか覚えていないが、たぶん、整数リテラルが多相だったので、だと思う。他の条件も要見直し。

本件は、クローズする（最後の修正、コメントアウトでなく削除してしまおう）。

- arithmeticseq.hs -> sample175.hs
- wheresample.hs -> sample176.hs
- aseq.hs は削除