095: 型チェックの方式変更（正常化）

↑ up

- issued: 2020-05-25
- 分類： C 改善項目
- status: Closed (2020-05-26)

概要

073 の対処で、DictPass にて Pred を補う処理をいれたが、なんだか混乱した実装になっており、やたら @@ の呼び出し時間が伸びている原因にもなっているよう。

このあたりを改善したい。

調査ログ

2020-05-23 (Sat)

Typing をみなおしてみると、ptype2.hs などの p に Pred がつかないのは、monomorphism restriction 処理によるもののようだった。仮に、この処理を disable にしてみると、p は期待通り Pred がついた状態のまま TrCore に処理がわたる。

となると、DictPass で Pred を補う理由は _fail だけになるが、こちらは、TrCore でより素直に型を補うことができる。

たとえば、

```
gcd :: (Integral a) => a -> a -> a
gcd 0 0 = rhs1
gcd x y = rhs2
```

というような定義において、この右辺を変換するときの _fail の型は、gcd の型と、左辺の引数の個数から静的に `(Integral a) => a` とわかる。そうして、TrCore の結果（core0）の型に、すべて正しく Pred がつけば、DictPass で Pred を補う処理は不要になる。

また、073 の過程で、BindGroup に悩んで、Rename で Let をばらすような処理も仮にいれてあったが、あれも一旦は不要になるのでとる。

おおむねこのような変更を実施。

以下、変更した各箇所にコメントしていく。git diff で出てきた順なので、変更した順番にはなっていない。

以下は、DictPass にて Pred を補う処理を削除する変更。以前に戻しただけ：

```
--- a/compiler/src/DictPass.hs
```

```
+++ b/compiler/src/DictPass.hs
@@ -46,33 +46,13 @@ tcBind (Rec bs) ce maybest = tbloop st0 bs []
     tcbind (v@(TermVar n qt@(qs :=> t)), e) st
       | isOVExpr e = ((v, e), st)
       | otherwise =
-        let pss' = tcPss st
-
-            (qtr@(qsr :=> tr), st2) = runState (getTy e) st
-            qtr'@(qsr' :=> _)  = normQTy qtr
-
-            cnd = case qt of
-              ([] :=> (TVar _)) -> not (null (tv qsr'))
-              _                 -> False
-
-            qt'@(qs' :=> t') = if cnd
-                  then qtr'
-                  else qt
-
-            pss = (zip qs' (repeat n))
-
-            rqty0 = tcReplaceQty st2
-            rqty = if cnd
-                   then [(n, qtr')]
-                   else []
-
-            v' = (TermVar n qt')
-
-            st' = st2{tcPss=(pss++pss'), tcReplaceQty=rqty0++rqty}
-            (e', st'') = runState (tcExpr e qt') st'
+        let pss = (zip qs (repeat n))
+            pss' = tcPss st
+            st' = st{tcPss=(pss++pss')}
+            (e', st'') = runState (tcExpr e qt) st'
         in
-          if null qs' then ((v', e'), st''{tcPss=pss'})
-          else ((v', Lam (mkVs n qs') e'), st''{tcPss=pss'})
+          if null qs then ((v, e'), st''{tcPss=pss'})
+          else ((v, Lam (mkVs n qs) e'), st''{tcPss=pss'})
     tcbind _ _ = error "tcbind: must not occur."
```

次は、Rename にて、BindGroup が最小になっていないときに、bind をひとつずつばらして Let をネストさせていた処理を取り除いたもの。取り除いても、テストで pass/fail する数はかわらなかった：

```
--- a/compiler/src/Rename.hs
+++ b/compiler/src/Rename.hs
@@ -657,7 +657,7 @@ renRhs (A.UnguardedRhs e ds) =

 renRhs (A.GuardedRhs gs decls) =
   let eFail = A.VarExp (Name {origName="Prim.FAIL", namePos=(0,0), isConName = False})
-      fail_local = A.VarExp (Name {origName="_fail", namePos=(0,0), isConName = False})
+      fail_local = A.VarExp (Name {origName="_fail#", namePos=(0,0), isConName = False})
       cnvGs []                          = fail_local
       -- todo cnvGs support most simple case that has only one statement.
       cnvGs (([A.ExpStmt e1], e2):gs') = A.IfExp e1 e2 (cnvGs gs')
@@ -705,12 +705,7 @@ renExp (A.LetExp ds e) = do
   e' <- renExp e
   exitLevel
   let bgs = toBg tbs
-      (es, iss) = head bgs
-      f []        bdy = bdy
-      f (is:iss') bdy = Let ([],[is]) (f iss' bdy)
-  if null es
-    then return $ f iss e'
-    else return (Let (head bgs) e')
+  return (Let (head bgs) e')
```

つぎに、TrCore において、_fail の型を差し替える処理。原理は上述のとおりだが、実装はいい加減すぎるかもしれない。

あきらかに問題だなと気づきつつ放置しているのは、推定した _fail の述語が過剰なケース（引数の数に応じて型を小さくした結果いらなくなったものを削除していない）への対応。

```
diff --git a/compiler/src/TrCore.hs b/compiler/src/TrCore.hs
index 1e247b0..061005c 100644
--- a/compiler/src/TrCore.hs
+++ b/compiler/src/TrCore.hs
@@ -12,7 +12,9 @@ import qualified Typing                      as Ty (Expr (..), Literal (..),
                                                     Pat (..))
```

```
 import           Control.Monad.State.Strict
+import           Data.List.Split            (splitOn)
 import qualified Data.Map.Strict            as Map
+import           Data.Maybe
 import           Debug.Trace

 ptypes :: Type -> [Type]
@@ -36,10 +38,15 @@ data TrcState = TrcState { trcAs     :: Assumps
                          , trcBstack :: [Bind]
                          , trcNum    :: Int
                          , trcConsts :: ConstructorInfo
+                         , trcFailSc :: Scheme
                          }

 type TRC a = State TrcState a

+putFailSc :: Scheme -> TRC ()
+putFailSc sc = do st <- get
+                  put st{trcFailSc = sc}
+
 getCi :: TRC ConstructorInfo
 getCi = do
   st <- get
@@ -62,7 +69,7 @@ translateVdefs ::
   Assumps -> [(Id, Pat.Expression)] -> ConstructorInfo -> Bind
 translateVdefs as vdefs ci =
   let
-    st = TrcState as (Rec []) [] 0 ci
+    st = TrcState as (Rec []) [] 0 ci (Forall [] ([] :=> tUnit)) -- tUnit is dummy
     (_, st') = runState (transVdefs vdefs) st
   in
    trcBind st'
@@ -91,9 +98,13 @@ appendAs as' = do
 appendBind :: (Id, Expr) -> TRC ()
 appendBind (n, e) = do
   t <- typeLookup n
+  scfail <- trcFailSc <$> get
+  tfail <- freshInst' scfail
+  let isfail = last (splitOn "." n) == "_fail#"
+      t' = if isfail then tfail else t
```

```
   st <- get -- this must be after the typeLookup above.
   let Rec bs = trcBind st
-      bs' = bs ++ [(TermVar n t, e)]
+      bs' = bs ++ [(TermVar n t', e)]
   put st{trcBind = Rec bs'}

 pushBind :: TRC ()
@@ -112,11 +123,19 @@ popBind = do
   put st'
   return $ trcBind st

+calcFailSc :: Scheme -> [a] -> Scheme
+calcFailSc sc [] = sc -- todo: simplify ks and ps
+calcFailSc (Forall ks
+            (ps :=> TAp (TAp (TCon (Tycon "(->)" (Kfun Star (Kfun Star Star)))) _) t'))
+           (a:as)
+  = calcFailSc (Forall ks (ps :=> t')) as
+
 transVdef :: (Id, Pat.Expression) -> TRC ()
 transVdef (n, Pat.Lambda ns expr) = do
   as <- getAs
   sc <- find n as
   qt <- freshInst' sc
+  let failsc = calcFailSc sc ns
   let ts = case qt of
         (_ :=> t') -> ptypes t'
       vs = zipWith f ns ts
@@ -125,8 +144,15 @@ transVdef (n, Pat.Lambda ns expr) = do
       f n' t' = let qf' = filter (\pr -> head (tv pr) `elem` tv t') qf
                 in TermVar n' (qf' :=> t')
       as' = Map.fromList [(n', Forall [] (qf' :=> t')) | TermVar n' (qf' :=> t') <- vs]
+
   appendAs as'
+
+  failsc_save <- trcFailSc <$> get
+  putFailSc failsc
   expr' <- transExpr expr
+
+  st <- get
+  put st{trcFailSc=failsc_save}
```

```
   appendBind (n, lam' vs expr')
   putAs as
   where
@@ -134,6 +160,9 @@ transVdef (n, Pat.Lambda ns expr) = do
     lam' vs e = Lam vs e

 transVdef (n, e) = do
+  as <- getAs
+  sc <- find n as
+  putFailSc sc
   expr' <- transExpr e
   appendBind (n, expr')

@@ -155,8 +184,6 @@ transExpr (Pat.Case n cs) = do
   return $ Case scrut case_bndr alts
   where
     trClauses [] alts = do
-      -- defAlt <- mkDefAlt
-      -- return (defAlt:alts)
       return alts
     trClauses (Pat.Clause (i :>: scm) ns expr : cs') alts = do
       qt <- freshInst' scm
@@ -174,12 +201,6 @@ transExpr (Pat.Case n cs) = do
       -- TODO: qt is the type of the constructor. Should it be the type of lhs?
       let alt = (DataAlt (DataCon i vs qt), [], expr')
       trClauses cs' (alt:alts)
-    {-
-    mkDefAlt = do
-      a <- newTVar' Star
-      e' <- transExpr e
-      return (DEFAULT, [TermVar n ([] :=> a)], e')
-    -}

 transExpr (Pat.Fatbar (Pat.OtherExpression e) f) = do
   f' <- transExpr f
```

最後に、Typing にて、mono morphism restriction を抑止した箇所。トップレベル binding についてのみ発動するように改造すべきなのかもしれないが、いまのところ単純に常に disable している。

```
diff --git a/compiler/src/Typing.hs b/compiler/src/Typing.hs
index 722c128..b555a8a 100644
--- a/compiler/src/Typing.hs
+++ b/compiler/src/Typing.hs
@@ -606,7 +606,7 @@ tiImpls ce as bs = do ts <- mapM (\_ -> newTVar Star) bs
                           vss = map tv ts'
                           gs  = foldr1 union vss \\ fs
                       (ds, rs) <- split ce fs (foldr1 intersect vss) ps'
-                      if restricted bs then
+                      if False && restricted bs then
                         let gs'  = gs \\ tv rs
                             scs' = map (quantify gs' . ([]:=>)) ts'
                         in return (ds ++ rs, Map.fromList (zip is scs'))
```

メモ

この変更により、以下の test/samples は通らなくなった：

- sample171.hs
- sample217.hs
- sample224.hs
- sample225.hs

逆に、以下は通るようになった。

- strlenx.hs
- divquot2.hs

なお、この修正の前後で、実行時間は縮んでいる。

修正前 (sample152.mod.hs)：

- 36.552s
- 36.481s
- 36.667s

修正後：

- 28.032s
- 27.528s
- 27.606s

まだ、DictPass での Subst は過剰だと思われる（複数の束縛間で共有する必要のない Subst を状態にため込んで肥大化しているように思う）ので、これを削減して速度の変化をみたい（これは 094 の継続課題）。