

ST výpočty se stavem

IB016 Seminář z funkcionálního programování

Vladimír Štill, Martin Ukrop

Fakulta informatiky, Masarykova univerzita

Jaro 2015

Dosud dva typy funkcí v Haskellu

- čisté, nemají žádný stav
- IO, mohou libovolně měnit stav systému (disk, paměť, síť)
 - umožňuje vstup a výstup
 - a měnitelný stav v paměti (IORef, IO pole)
 - ale nelze bezpečně opustit IO monádu

Existuje něco mezi tím?

Dosud dva typy funkcí v Haskellu

- čisté, nemají žádný stav
- IO, mohou libovolně měnit stav systému (disk, paměť, síť)
 - umožňuje vstup a výstup
 - a měnitelný stav v paměti (IORef, IO pole)
 - ale nelze bezpečně opustit IO monádu

Existuje něco mezi tím?

- možnost pracovat se stavem (paměť), ale ne se vstupem a výstupem (disk, síť)
- výpočty, které vnitřně používají stav/modifikují proměnné, ale navenek referenčně transparentní
- možnost získat čistý výsledek výpočtu

State Transformers (ST)

State Threads/State Transformers: `Control.Monad.ST`.

- funkce v ST může vnitřně používat stav, ale navenek být referenčně transparentní
- typovým systémem garantovaná bezpečnost
- ST s je monáda, tedy lze řetězit pomocí (`>>=`) a do

Hodnota (typu `a`) se stavem: `ST s a`

- stav obsahuje mapování referencí (lokací proměnných proměnných) na hodnoty
- měnitelné proměnné: `Data.STRef`
- měnitelná pole: `Data.Array.ST`

```
foo :: Int -> ST s Int
foo x = do
  ref <- newSTRef x
  modifySTRef ref (+ 1)
  readSTRef ref
```

```
>λ= foo 1
<<ST action>>
>λ= runST (foo 1)
2
>λ= :t runST $ foo 1
runST $ foo 1 :: Int
```

Měnitelné reference: Data.STRef

- `data STRef s a`
reference na typ `a`
- `newSTRef :: a -> ST a (STRef s a)`
vytvoří novou měnitelnou proměnnou ve stavu `a` a vrátí referenci na ni
- `readSTRef :: STRef s a -> ST s a`
nemění stav, jen z něj přečte hodnotu
- `writeSTRef :: STRef s a -> a -> ST s ()`
změní hodnotu ve stavu
- `modifySTRef :: STRef s a -> (a -> a) -> ST s ()`
líně aplikuje funkci na proměnnou odpovídající referenci (funkce se aplikuje až v okamžiku čtení reference)
- `modifySTRef' :: STRef s a -> (a -> a) -> ST s ()`
striktní modifikace

Potřebujeme spustit sekvenci ST akcí a získat čistý výsledek.

- spuštění musí začít ve stavu v němž nejsou alokovány žádné proměnné
- nesmí být možné použít proměnnou z jiné sekvence akcí
- myšlenka: ST je indexován typem stavu, bezpečně spustit lze, pokud může být stav libovolný

Spouštění ST akcí

Potřebujeme spustit sekvenci ST akcí a získat čistý výsledek.

- spuštění musí začít ve stavu v němž nejsou alokovány žádné proměnné
- nesmí být možné použít proměnnou z jiné sekvence akcí
- myšlenka: ST je indexován typem stavu, bezpečně spustit lze, pokud může být stav libovolný

`runST :: (forall s. ST s a) -> a`

- `forall s.` v typu říká, že parametr `runST` musí být polymorfní v `s`

Spouštění ST akcí

Potřebujeme spustit sekvenci ST akcí a získat čistý výsledek.

- spuštění musí začít ve stavu v němž nejsou alokovány žádné proměnné
- nesmí být možné použít proměnnou z jiné sekvence akcí
- myšlenka: ST je indexován typem stavu, bezpečně spustit lze, pokud může být stav libovolný

`runST :: (forall s. ST s a) -> a`

- `forall s.` v typu říká, že parametr `runST` musí být polymorfní v `s`
- nejde např. `runST (newSTRef 0)`, a nemůže záviset na `s`

Pole v ST: Data.Array.ST

```
data STArray s i e
runSTArray :: Ix i =>
  (forall s. ST s (STArray s i e)) -> Array i e
```

- pole indexované typem i , hodnoty typu e
- měnitelné pole v ST s monádě
- rozhraní k poli dáno modulem `Data.Array.MArray`¹
- `getBounds :: Ix i => STArray s i e -> ST s (i, i)`
- `newArray :: Ix i => (i, i) -> e`
`-> ST s (STArray s i e)`
inicializuje všechny prvky danou hodnotou
- `readArray :: Ix i => STArray s i e -> i -> ST s e`
- `writeArray :: Ix i => STArray s i e -> i -> e`
`-> ST s ()`
- `freeze :: Ix i => STArray s i e -> ST s (Array i e)`
- `thaw :: Ix i => Array i e -> ST s (STArray s i e)`

¹typy zkonkretizovány

IO je speciální případ ST: `IO a ~ ST RealWorld a`

- „RealWorld is deeply magical.“²
- `stToIO :: ST RealWorld a -> IO a`
- hodnoty typu `ST RealWorld a` nelze spustit pomocí `runST`

²Oficiální dokumentace `Control.Monad.ST`

IO je speciální případ ST: $IO\ a \sim ST\ RealWorld\ a$

- „RealWorld is deeply magical.“²
- $stToIO :: ST\ RealWorld\ a \rightarrow IO\ a$
- hodnoty typu $ST\ RealWorld\ a$ nelze spustit pomocí $runST$

ST bylo zavedeno v článku Lazy Functional State Threads (John Launchbury, Simon L Peyton Jones, 1994), který je poměrně dobře čitelný (vyjma formální definice sémantiky).

²Oficiální dokumentace `Control.Monad.ST`

```
-- | apply function to value at given index of array
modifyArray :: Ix i => STArray s i e -> i
              -> (e -> e) -> ST s ()

-- | replace each element of array with sum of
--   values up to this element (inclusive)
sumArray :: (Ix i, Num n) => STArray s i n -> ST s ()
```