Reasoning and Derivation of Monadic Programs

Shin-Cheng Mu
Tom Schrijvers
Koen Pauwels

Content

- Equational reasoning
- Reasoning with monads
- Goal of the paper
- Our contribution

(a + b) * (a - b)

[distr: forall x y z. x * (y + z) = x*y + x*z]

(a + b)*a - (a + b)*b

[distr: forall x y z. x * (y + z) = x*y + x*z]

a^2 + b*a - a*b - b^2

[comm: forall x y. x*y = y*x]

a^2 + a*b - a*b - b^2

[addinv: forall x. x - x = 0]

a^2 - b^2

```python
def increment(n):
    print(“surprise!”)
    return n+1

four = increment(3)

print(four + four)
```

```
def increment(n):
    print(“surprise!”)
    return n+1


print(increment(3) + increment(3))
```

```haskell
increment :: Int -> IO Int
increment n = do putStrLn "surprise!"
                 return (n+1)


main = let four = increment 3
       in  print $ four + four  -- error!
```

```haskell
increment :: Int -> IO Int
increment n = do putStrLn "surprise!"
                 return (n+1)


main = do four <- increment 3
          print $ four + four
```

```haskell
increment :: Int -> IO Int
increment n = do putStrLn "surprise!"
                 return (n+1)


main = let four = increment 3
       in  ((+) <$> four <*> four)
            >>= print

     = ((+) <$> increment 3 <*> increment 3)
     >>= print
```

```
class Monad m => MonadState s m | m → s where
  get :: m s
  put :: s → m ()


put s >> put s’ = put s’

put s >> get    = put s >> return s

get >>= put     = skip

get >>= \s -> get >>= \s’ -> k s s’
                = get >>= \s -> k s s
```

```
queens :: MonadNondet m => Int -> m [Int]
queens n = perm [0 .. n-1] >>= assert safe
```

```
class Monad m => MonadNondet m where
  fail :: m a
  (||) :: m a -> m a -> m a
```

associativity:

```
(m || n) || k = m || (n || k)
```

zero element for (||):

```
fail || m = m = m || fail
```

distributivity from left:

```
(m || n) >>= f = (m >>= f) || (n >>= f)
```

left zero for (>>=):

```
fail >>= f = fail
```

```
queens :: MonadNondet m => Int -> m [Int]
queens n = perm [0 .. n-1] >>= assert safe
```

[0,1,…]

(n-2)!

```
class (MonadNondet m, MonadState s m) =>
      MonadNDState s m
```

right-distributivity:

```
  m >>= (\x -> f1 x || f2 x) =
  (m >>= f1) || (m >>= f2)
```

right-zero:

```
  m >> fail                       = fail
```

```haskell
queens :: (MonadNondet m, MonadState (Int,[Int],[Int]))
       => Int -> m [Int]
queens n = protect (put (0,[],[]) >> queensBody [0 .. n-1]



queensBody :: (MonadNondet m, MonadState (Int,[Int],[Int]))
           => [Int] -> m [Int]
queensBody [] = return []
queensBody xs = do (x,ys) <- select xs
                   st     <- get
                   guard (check st x))
                   put (f st x)
                   fmap (x:) (queensBody ys)
  where ...
```

```
class (MonadNondet m, MonadState s m) =>
      MonadNDState s m
```

right-distributivity:

```
  m >>= (\x -> f1 x || f2 x) =
  (m >>= f1) || (m >>= f2)
```

right-zero:

```
  m >> fail                       = fail
```

```
modify next >> search >>= modReturn prev
  where
    modify f      = get >>= (put · f)
    modReturn f v = modify f >> return v
```

rollback unreachable:

```
  modify next >> fail >>= modReturn prev
```

rollback executed multiple times:

```
  modify next >>  (m1 || m2 || m3)
                >>= modReturn prev
```

```
side (modify next)
  || m1
  || m2
  || m3
  || side (modify prev)

where
  side m = m >> fail
```

```
putR :: (MonadNondet m, MonadState s m)
     => s -> m ()

putR s =
  get >>= \s0 -> put s || side (put s0)
```

```
putR s >> comp

                                          [def putR]

(get >>= \s0 -> put s || side (put s0)) >> comp

                  [assoc monad law, left distributivity]

get >>= \s0 -> (put s          >> comp)
            || (side (put s0) >> comp)

                                         [left zero]

get >>= \s0 -> (put s >> comp) || side (put s0)
```

get-put:

```
  get >>= putR = return ()


(get >>= putR) >> put t
  = get >>= \s -> put t || side (put s)

return () >> put t
  = put t
```

Replace all occurrences of `put` by `putR`

$$m =_{GS} n$$

$$\iff$$

forall C. run(C[m]) = run(C[n])

$$m =_{LS} n$$

$$\Longleftrightarrow$$

$$\forall C.\ \mathrm{run}(\mathrm{trans}(C[m])) = \mathrm{run}(\mathrm{trans}(C[n]))$$

get-get:

`get (\s1 -> get(\s2 -> k s1 s2))` $=_{LS}$ `get (\s -> k s s)`

put-get:

`putR x (get k)` $=_{LS}$ `putR x (k x)`

get-put:

`get (\s -> putR s k)` $=_{LS}$ `k`

put-put:

`putR x (putR y m)` $=_{LS}$ `putR y m`

right-distributivity:

$$\texttt{putR x m1 || putR x m2} =_{LS} \texttt{putR x (m1 || m2)}$$

right-zero:

$$\texttt{putR x fail} =_{LS} \texttt{fail}$$