sligt.wordpress.com

Testing c using QuickCheck.

QuickCheck is a nice tool for testing Haskell code. But imagine testing c code against a reference haskell implementation! I'll define a FFI binding between c and Haskell then test the c code against a Haskell reference implementation using QuickCheck.

The post is organized as follows

- The c code
- The Haskell FFI binding
- The Haskell Test code

The c code is as follows.

```
/* smallfield.h defines the operations implemented in F_3^97 */
// an element of F_3^97 is represented as an arrray of 26 bytes.
// 13 bytes for the low bits and 13 bytes for the high bits.

#define BYTEARRSIZE 13

typedef struct sfe { // small_field_element
  char low[BYTEARRSIZE];
  char high[BYTEARRSIZE];
} sfe;

void sf_add(sfe *a, sfe *b, sfe *o);


/* smallfield.c */
#include "smallfield.h"

void sf_add(sfe *a, sfe *b, sfe *o) {
  int i;
  for (i = 0; i < BYTEARRSIZE; i++) {     o->low[i] =
      ((a->low[i] ^ b->low[i]) & (~b->high[i]) & (~a->high[i]))
      | (a->high[i] & b->high[i]);
    o->high[i] =
      ((~(a->high[i] | a->low[i])) & b->high[i])
```

```
      | ((~a->high[i] & a->low[i]) & b->low[i])
      | (a->high[i] & ~(b->high[i] | b->low[i]));
  }
}
```

The function sf_add looks very suspicious but it's a direct implementation of the boolean formula I wrote on the whiteboard so it is definitely correct!

But even that I know it's correct because I made it, some evidence for the correctness would be great. I'll check it against a haskell implementation of the same extension field. The first thing is to define a FFI bridge between c and haskell. The tool hsc2hs can be used for this task. The following file is the definition of the hsc2hs FFI interface.

```
{- SmallField.hsc -}
{-# LANGUAGE ForeignFunctionInterface #-}

module SmallField where

import Foreign
import Foreign.C.Types
import Foreign.Marshal.Array()

#include "smallfield.h"

data SFE = SFE {low:: [CChar], high :: [CChar]} deriving
(Show,Read,Eq)
type SFEPtr = Ptr SFE

foreign import ccall "static smallfield.h sf_add"
  f_sf_add :: SFEPtr -> SFEPtr -> SFEPtr -> IO ()

smallFieldAdd :: SFE -> SFE -> SFE
smallFieldAdd a b = unsafePerformIO $
  do [ap,bp,resp] <- mapM new [a,b,SFE [] []]
     f_sf_add ap bp resp
     elm <- peek resp
     mapM free [ap,bp,resp]
     return elm

instance Storable SFE where
  sizeOf _ = (#size sfe)
  alignment _ = alignment (undefined :: CChar)
  peek ptr = do
    l <- (peekArray 13 (lowBitsAddr ptr))
```

```
    h <- (peekArray 13 (highBitsAddr ptr))
    return $ SFE l h

  poke ptr (SFE lowBits highBits) = do
    pokeArray (lowBitsAddr ptr) lowBits
    pokeArray (highBitsAddr ptr) highBits

lowBitsAddr = (#ptr sfe, low)
highBitsAddr = (#ptr sfe, high)
```

Now lets make the test code for sf_add in haskell.

```
{- Test.hs -}
{-# LANGUAGE EmptyDataDecls, TypeSynonymInstances,
MultiParamTypeClasses #-}

import SmallField

import Math.CAA.PrimeField
import Math.CAA.ExtensionField
import Math.CAA.Polynomial

import System.Random
import Test.QuickCheck

import Foreign.C.Types

-- Definition of field F_3
data PrimeNumber3
instance PrimeAsType PrimeNumber3 where
  primeValue _ = 3

type FF3 = PF PrimeNumber3
-- End of F_3

-- Definition of the extension field F_{3^97} with the reduction
-- polynomial x^97+x^12-1
data ReductionPolynomial
instance PolynomialAsType FF3 ReductionPolynomial where
  pvalue _ = (fromCoefficients [(one,97),(one,12),(addInv one,0)])
:: UP FF3

type SmallField = ExtField FF3 ReductionPolynomial
-- End of extension field definition
```

```
-- Property: addition in the c code and haskell code gives the
same
-- result.
prop_addition :: SmallField -> SmallField -> Bool
prop_addition a b =
  smallFieldAdd (elmToSFE a) (elmToSFE b) == (elmToSFE (a<+>b))

main =
  do quickCheck prop_addition

-- QuickCheck arbitrary instance for the type SmallField.
instance Arbitrary SmallField where
  arbitrary = elements $ take 100 $ randoms (mkStdGen 42)

-- Helper functions to convert a type SmallField to the c function
-- representation.
elmToSFE :: SmallField -> SFE
elmToSFE (ExtField a) = SFE lowBits highBits
  where coeffs = toCoefficientList a
        lowBits = compresser $ map (\x -> if x == one then True
else False) coeffs
        highBits = compresser $ map (\x -> if x == addInv one then
True else False) coeffs

compresser :: [Bool] -> [CChar]
compresser as = take 13 $(compress as) ++ (repeat 0)
  where compress [] = []
        compress as = (convertToCChar (take 8 as)):(compresser
(drop 8 as))

convertToCChar [] = 1
convertToCChar as = sum $ map (\(a,i) -> (if a == True then 1 else
0)*2^i) (zip as [0..])
```

Compile and run the code to see it works.

```
hsc2hs SmallField.hsc
gcc -c -o smallfield.o smallfield.c
ghc --make Test.hs smallfield.o

$ ./Test
+++ OK, passed 100 tests.
```

This entry was posted on May 28, 2010 at 1:41 pm and is filed under Haskell.