

## *Software testing: week 3 report*

*Tuba Kaya Chomette, Sander Leer, Martijn Stegeman*

*September 29, 2013*

```
module Week3Sol
```

```
where
```

```
import Techniques
import Week3
import Week2Solutions
```

```
import System.Random
import Control.Monad
import Data.List
import Data.Char
```

### *getIntList*

The type of this function provides no direct way to limit the amount of random numbers to generate. In order to be able to generate a usable infinite list of numbers, we have to do it lazily.

```
getIntList :: IO [Int]
getIntList = do
    r <- newStdGen
    return (randomRs (0,15) r)

randomInts :: Int -> IO [Int]
randomInts n = liftM (take n) getIntList
```

### *isPermutation*

```
isPermutation :: Eq a => [a] -> [a] -> Bool
isPermutation xs ys = xs `elem` permutations ys
```

### *Random testing for isPermutation*

This function generates a random list and checks if all permutations (according to the standard permutations function) are actually permutations.

```
randomTestPerm :: IO Bool
randomTestPerm = do
```

```
x <- randomInts 3
return (and (map (isPermutation x) (drop 1 (permutations x))))
```

This function generate a random list, adds an extra number in front or at the back, and verifies that they are not permutations of the original.

```
randomTestPerm2 :: IO Bool
randomTestPerm2 = do
  x <- randomInts 3
  return ((not (isPermutation x (0:x))) && (not (isPermutation x (x++[0]))))
```

### *Testing the Cnf converter*

```
checkCNFs :: IO ()
checkCNFs = testForms 50 checkCnf
```

All randomly generated formulas out of `testForms` pass as correct when converted to CNF formulas. We do notice, however, that the randomly generated formulas also sometimes contain disjunctions or conjunctions without any arguments (e.g. `+( )` or `*( )`). That is to say: an AND and OR operators without arguments. This should possibly not be a valid formula in logic (or valid CNF). The origin of this problem lies within the random formula generator; it generates conjunctions and disjunctions with 0.5 arguments. This can obviously be [2..] in order to conform more to the usual form of logical formulas.

```
getRandomBool :: IO Bool
getRandomBool = do
  n <- getRandomInt 1
  return (n == 1)

getRandomAtomName :: IO String
getRandomAtomName = do
  n <- getRandomInt 25
  return [chr (n + ord 'A')]

getRandomTermName :: IO String
getRandomTermName = do
  n <- getRandomInt 25
  return [chr (n + ord 'a')]

getRandomTerm :: IO Term
getRandomTerm = do
  b <- getRandomBool
```

```

s <- getRandomTermName
if b then do
  return (V s)
else do
  t <- getRandomTerms 2
  return (F s t)

getRandomTerms :: Int -> IO [Term]
getRandomTerms 0 = return []
getRandomTerms d = do
  t <- getRandomTerm
  ts <- getRandomTerms (d-1)
  return (t:ts)

getRandomFormula :: Int -> IO Formula
getRandomFormula 0 = do
  s <- getRandomAtomName
  t <- getRandomTerms 2
  return (Atom s t)

getRandomFormula d = do
  n <- getRandomInt 8
  case n of
    0 -> do
      s <- getRandomAtomName
      t <- getRandomTerms 2
      return (Atom s t)
    1 -> do
      t1 <- getRandomTerm
      t2 <- getRandomTerm
      return (Eq t1 t2)
    2 -> do
      f <- getRandomFormula (d-1)
      return (Neg f)
    3 -> do
      f1 <- getRandomFormula (d-1)
      f2 <- getRandomFormula (d-1)
      return (Impl f1 f2)
    4 -> do
      f1 <- getRandomFormula (d-1)
      f2 <- getRandomFormula (d-1)
      return (Equi f1 f2)
    5 -> do
      m <- getRandomInt 5

```

```

        fs <- getRandomFormulas (d-1) m
        return (Conj fs)
6 -> do
    m <- getRandomInt 5
    fs <- getRandomFormulas (d-1) m
    return (Disj fs)
7 -> do
    s <- getRandomTermName
    f <- getRandomFormula (d-1)
    return (Forall s f)
8 -> do
    s <- getRandomTermName
    f <- getRandomFormula (d-1)
    return (Exists s f)

getRandomFormulas :: Int -> Int -> IO [Formula]
getRandomFormulas _ 0 = return []
getRandomFormulas d n = do
    f <- getRandomFormula d
    fs <- getRandomFormulas d (n-1)
    return (f:fs)

getRandomFrms :: IO Formula
getRandomFrms = do
    d <- getRandomInt 2
    getRandomFormula d

```