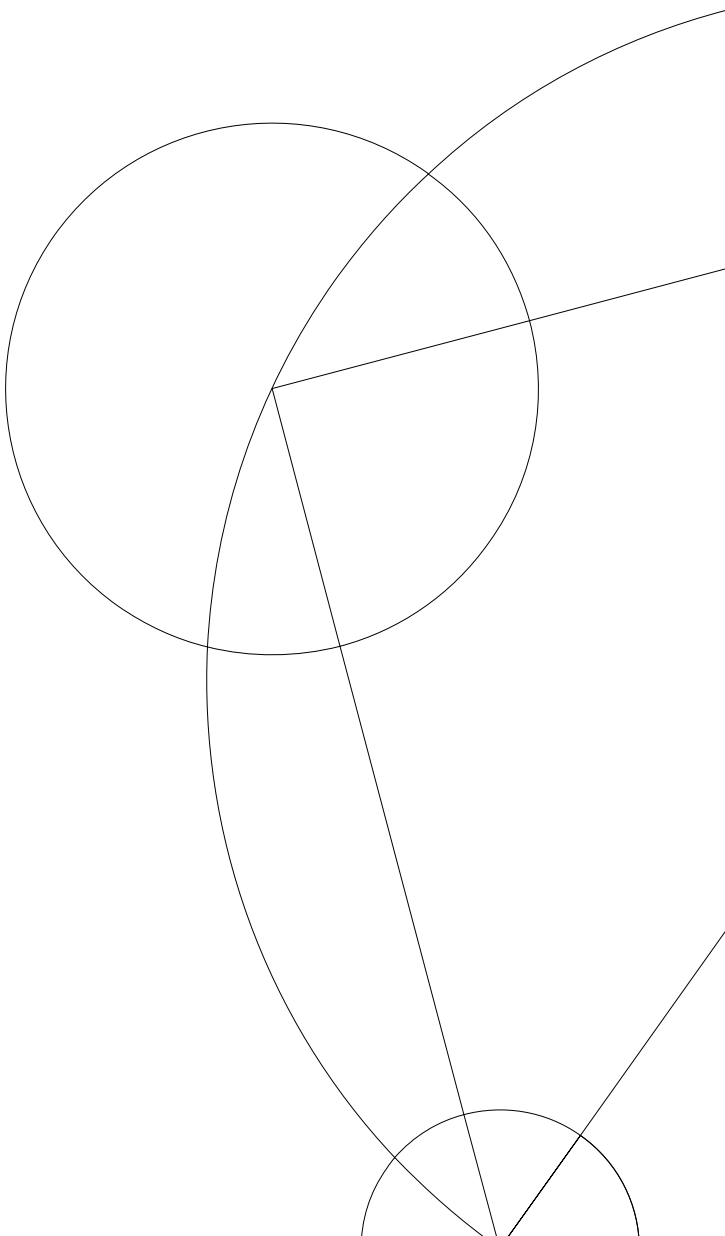
KØBENHAVNS UNIVERSITET

Home Assignment 5

QuickCheck

qbp758 / zxf339

AP22 Assignment 5 Group 8



Date: 15. oktober 2022

Part 1: Haskell

Regarding our generators, firstly, we use lowercase letters(from a to z) to generate a 'Ident' string . Secondly, we use 'shrink' instance to limit the generation of long expressions. To work with the 'simplifier' function, the 'Let' expression generation will not contain 'Var' expressions. The code snippet is as follows.

```haskell
--  Lowercase letter generation
genSafeChar :: Gen Char
genSafeChar = elements ['a'..'z']
-- Ident generation
arbitraryIdent :: Gen String
arbitraryIdent = listOf1 genSafeChar

-- helper function
-- for generate the `Let` expressions do not contain `Var` key word
isVarInLet :: Expr -> Bool
isVarInLet e = not ( "Var" `isInfixOf` show e )

-- Let expression generation
do
    i <- arbitraryIdent
    x <- suchThat (exprN(n `div` 2)) isVarInLet -- not contain `Var`
    y <- exprN(n `div` 2)
    return $ Let i x y
```

Regarding the quality of our generators, we use 'collect' function to print logs of different generations. The log snippet is as follows. We put the full log save as a file which is available in Absalon and the file name is 'quality of my generation.txt'

```
QuickCheck tests
   Evaluating a simplified expression does not change its meaning: OK
     +++ OK, passed 100 tests:
      2% Const 0
      1% Const (-1)
      1% Const (-10)
      1% Const (-12)
      1% Const (-13)
      1% Const (-23)
      1% Const (-28)
      1% Const (-4)
      1% Const (-55)
      ...
```

Part 2: Erlang

We implemented shrinking, because it will easy to find bugs. We use '?LETSHRINK' to limit big BSTs in the code. The code snippet is as follows.

```erlang
bst(Key, Value) ->
    ?LAZY(
        ?LETSHRINK(KVS, eqc_gen:list({Key, Value}),
            {call, lists, foldl, [
                fun({K,V}, T) -> insert(K, V, T) end,
                {call, bst, empty, []},
                KVS]}
            )
    ).
```

Regarding the failing property and what bugs, we will give an example for you. In ‘noether‘ version, the test result shows some fails which is as follows. In this fail case, the key ‘e‘ is existed in a BST, then insert the same key ‘e‘ with different value to the BST, but the final result shows that the insert function is not work. So, in conclusion, the insert function has a bug in the ‘noether‘ version.

```
prop_insert_post: .....................Failed! After 70 tests.
{e,e,20,
   {call,lists,foldl,
         [#Fun<test_bst.28.127627523>,
          {call,bst,empty,[]},
          [{b,7},{h,3},{b,15},{e,-21},{c,7},{h,1},{g,-22}]]}}
   {found, -21} /= {found, 20}
```

Regarding the bugs of different versions, the summary bugs table are as follows.

Version	Bugs
noether	insert function is not update value when a key is existed at BST.
prelman	union function is not update value when a key is existed at BST.
protras	union function is not combine two BSTs by keys order
rhodes	union function is not combine two BSTs by keys order
robinson	delete function is just remove one when more than one same key.
snyder	delete function will remove all items.
wilson	• insert function is not update value when a key is existed at BST. • delete function is just remove one when more than one same key.
wing	insert function will add a new item and remove others existed items.

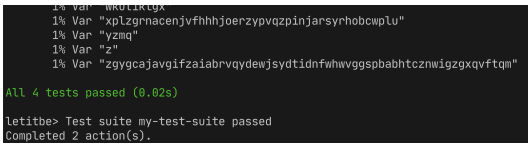
Part 3: Assessment of the Quality of Our code

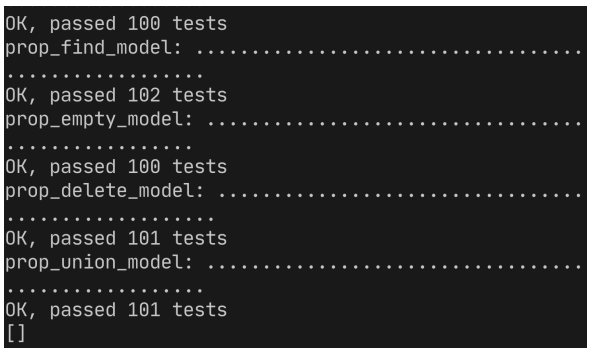
A. Completeness

We have completed all the questions, although we spent a bunch of time,but way less than last time.

B. Correctness

Our property test correctness is good and as shown as follow.





C. Efficiency

We did try our best to improve the efficiency of code operation and make efforts to improve space usage, and it turns out that our efforts paid off.

D. Maintainability

We both think the maintainability of our code is quite good. As we added enough common code to improve the readability of the code and abstracted functions to make our code logically clarified.