Functional Programming – Advanced Concepts and Applications
WS 2023/24

Till Mossakowski
Benjamin Junge

7. Excercise sheet

Issued: 2023-11-27
Due: 2023-12-04 & 2023-12-05

7.1 *Deque axioms*

Consider the double headed queue type from 6.2.

1. Formulate axioms (properties that should hold for all instances) on this type *(at least 6)* and explain why they should hold.
2. Turn your axioms into haskell predicates. Then use the module QuickCheck to test your predicates within the Deque module. Discuss the quickCheck results and potential problems that may arise from your predicates.

To be able to use QuickCheck you might have to download it by using `cabal update` followed by `cabal install QuickCheck` in a terminal/cmd prompt. Since quickCheck tests by generating test data, but only has predefined generators for types of the base module (and defining generators for custom types requires monadic operations) you need to use a Deque implementation that uses **type** `Deque a = [a]` instead of declaring a new type with the **data** keyword.

7.2 *More axioms: Sorted List*

Consider the sorted list type from 6.1. being implemented as **type** `SortedList a = [a]`

1. Which of the follow sentences are axioms for the sorted list type, which aren't and why? [1]
   Note that the axioms are not formulated in haskell syntax!
   - minimum(s) = head(s)   ∀(Sorted lists s)
   - maximum(s) = head(s)   ∀(Sorted lists s)
   - isSorted(x:s) = True   ∀(Sorted lists s and element x)
   - (x = head(s)) ⇒ length (s) = length (insert x s)   ∀(Sorted lists s and element x)
   - (x < head(s)) ⇒ (isSorted(x:s)) = True   ∀(Sorted lists s and element x)
   - (isSorted(x) ∧ isSorted(y)) ⇒ isSorted(x++y)   ∀(Sorted lists x and y)

   Can you think of a way to alter those sentences that aren't axioms in order to turn them into axioms?
2. Formulate the sentences of part one as haskell predicates and check them via quickCheck. The quantification does not have to be part of your predicate. *use **import** Data.List for easy access to a sort and insert function. You may have to define your own predicate isSorted.*

[1]: here refers to the build in function on lists while insert refers to the Sorted List specific counterpart

$\boxed{7.3}$ *Trace*

The module `Debug.Trace` allows for printing values from inside a function and is a useful tool for finding logic bugs.

1. Discuss the signature of the `trace` function and explain if trace is pure or not.
2. Use trace to figure out the order of recursive calls in an directly equivalent implementation of the fibonacci function fib, $fib(0) = 0, fib(1) = 1, fib(n) = fib(n-1) + fib(n-2)$

$\boxed{7.4}$ *Tasty*

Tasty is a testing framework for haskell that unifies modules like QuickCheck, HUnit, SimpleCheck etc. which are all used for testing. In order to use it use `cabal update` followed by `cabal install tasty` in a terminal/cmd prompt. Use the tasty framework to group the QuickCheck tests for Deque and/or SortedList and execute the testgroup. *For more information see Tasty.*

$\boxed{7.5}$ **Axiom puzzle*

The following set of axioms is formulated for the build-in list type. Figure out which basic build-in functions on lists (from Prelude) are represented by f_1 to f_6. Check your choices by utilizing QuickCheck. All axioms should hold simultaneously for your choice. a and b are arbitrary lists. [2]

1. a == $f_1 \circ f_1$ a
2. f_2 (f_1 b) (f_1 a) == f_1 (f_2 a b)
3. not (null a) ==> (a == f_3 (f_4 a) (f_5 a))
4. f_6 a + f_6 b == f_6 (f_2 a b)
5. not (null a) ==> (f_2 ($f_1 \circ f_5$ a) ([f_4 a]) == f_1 a)

Are there alternative function choices that could fulfil those axioms?

[2]Remember that infix operators can be used like normal function, e.g `(+) 1 2 == 1 + 2`