

# Lab 5 Software Specification and Testing

- **Don't forget** to follow the naming conventions as indicated in Lab 1.
  - Please indicate the time spent on each exercise
  - Add a comment in each Assignment file, that states how long you collectively worked on that assignment.
  - The comment should be: **-- Time spent: x minutes --** , with x as the amount of minutes it took.
- 

```
> import Data.List
> import Test.QuickCheck
> import Mutation
```

---

## Prerequisites:

- Take a look at the paper [FitSpec: refining property sets for functional testing](#).
  - Look at the example code in the *Mutation.hs* file.
  - Make a list of questions on specific points that cause difficulty of understanding.
  - Take a look back at [the weaker/stronger theory of the Hoare lab](#).
  - Look at the example code in the *FitSpec.hs* file and run the *main* function.
- 

In this lab, we write our own mutation testing framework like FitSpec. We manipulate the output data of a function under test, to check that any incorrect output for a given input fails the tests. This way, we can discover survivors, the minimal set of properties, and find all the conjectures.

For this lab, we use a 'multiplication table' implementation+properties as target for mutation testing. The related Haskell code can be found in *MultiplicationTable.hs*.

---

1. We provide some mutators to mutate the output of the list in *Mutation.hs*. Write down which types of output are not yet covered by these mutators, and about their weakness/strength. Come up with a list of other mutators and implement (a subset of) them.

**Hint:** A mutant can never be the same as the original output list.

**Deliverables:** List of mutators and rationale, implementation, indication of time spent.

---

2. Write a function `countSurvivors` that counts the number of survivors:

```
countSurvivors :: Integer -> [[[Integer] -> Integer -> Property]] -> (Integer ->
[Integer]) -> Integer
```

Where the first argument is the number of mutants (4000 in the FitSpec example).

And the second argument is the list of properties.

And the third argument is the function under test (the multiplication table function in this case).

And the output is the number of surviving mutants (0 in the FitSpec example).

Document the effect of which mutations are used and which properties are used on the number of survivors.

**Hint:** Consider the relation between properties and survivors.

**Another hint:** The above-mentioned function definition is not final. Feel free to modify it, for example by adding the mutations that should be used.

**Deliverables:** implementation, documentation of approach, effect of using different mutators/properties, indication of time spent.

---

3. Implement a function that calculates the minimal property subsets, given a 'function under test' and a set of properties.

**Deliverables:** implementation, documentation of approach, indication of time spent.

---

4. Implement a function that calculates the strength of a given set of properties, which is the percentage of mutants they kill.

**Deliverables:** implementation, documentation of approach, indication of time spent.

---

5. Implement function(s) that calculate the conjectures: properties that are equivalent, whose cases are subsets of other properties, etc.

**Deliverables:** implementations, documentation of approach, indication of time spent.

---

6. (1pt) **Bonus:** Create a function that we pass the function under test, a set of properties, and a number indicating the number of mutants, that visualizes the results of the functions from the previous exercises.

**Deliverables:** Implementation, indication of time spent.

---

Made with ❤️ and 🧑🏻 by Simon Baars