Advanced Programming (I00032)
A DSL for Model-Based Testing

Assignment 12

In the lecture we showed a simplified Model-Based test system based logical properties. The definitions of this system are available as the module `gastje`[1] on Blackboard[2]. In this assignment you extend this system and apply it.

1 Specific test-cases

The default interpretation for an argument of a property is universal quantification. In some situations you want to control the generation of test cases for an individual test. Implement an infix operator `For` that allows us to specify test-cases. For example:

```
pUpper :: Char → Bool
pUpper c = c ≠ toUpper c

Start = ["pUpper: ":test (pUpper For ['a'..'z'])]
```

Define an operator `For` and the associated instance of `prop`.

2 Input Selection

An other way to restrict the tests is by defining a precondition on test values. For instance:

```
Start = ["pUpper lower:": test (λc. isLower c ==> pUpper c)]
```

Extend the test system with the operator ==>. It is **not** required to count these rejected test cases separately as suggested in the lecture.

3 Tracing the Arguments of Equality

Many properties for testing contain an equality. For example:

```
pfac i = abs i < 10 ==> prod [1..i] == fac i
```

When testing finds a counterexample for such a property we typically want to know the values produced by the left- and righthand-side of the ==. Introduce an operator =.= which list the values of its arguments and behaves like the ==.

[1] A bug in the Clean compiler requires that you use the iTask environment for this module and that class constraints that involve generic functions are defined in a `.dcl`-module.

[2] There are some minor differences in identifiers and the way constraints are enforced to make this assignment easier.

4 Testing `Euro`

On Blackboard you find a module `cashModel` that defines a type `Euro` and a model for a cash register as discussed on slide 38 in the lecture.

Define properties for the operations (like `+`, `-`, and $\sim$) on type `Euro`. List the result of executing the associated tests as comment in your code.

5 Testing the Remove Action in the Model

The remove action, `Rem p`, for product `p` is the most complicated transition in the model. Write a property and test it to check whether the specified transition is fair. List the result of executing this test as comment in your code.

Deadline

The deadline for this assignment is December 14, 23:30h.