Advanced Programming (I00032) 2017
Generics by overloading & Clean Native Generics

Assignment 3

Goals of this exercise

In this exercise you learn how to implement and use the kind-indexed version of generic programming as well as native generic programming in Clean.

1 Kind-Index Generic Serialization

In the last exercise you implemented generic serialization. In the lecture we showed that the approach of exercise 2 fails when we need a kind different from `*`. The solution is to use kind-indexed generics as introduced in the lecture. Implement the class `serialize` based on the new version of generics. On blackboard you find a file `serialize3Start.icl` that contains all boilerplate definitions and some tests to check your implementation.

Your solution should ideally

1. avoid all generic information in the serialized form (the list of strings);
2. use only brackets for a constructor with arguments;
3. pass all tests.

The output of the test looks better if the program is compiled with `Basic Values Only` for Console in `Project Options...`. You can set this option in dialogue in the `Project` menu.

1.1 Tailor-made Serialization

It is possible to deviated from the generic route for specific types. For instance we can serialize the expression `(7,True)` as `["(","7",",","True",")"]`. Implement this for all 2-tuples while using the ordinary generic serialization for other types.

2 Serialization using Clean's Native Generics

Use the native generic from Clean to implement the class `serialize` defined as

```
class serialize a | read{|*|}, write{|*|} a
```

Use the same requirements and tailor-made version for tuples as above.

Do not forget to import `StdGeneric`. This will pass the required `-generics` flag to the compiler.

3 Generic Apply

On Blackboard you will find a file `genericMap` with the generic map function, `gMap`, used in the lecture. Use this to

1. apply the factorial function `fac` to all elements of tree `t`;
2. turn each integer `i` in list `l` to a tuple (`i`, `fac i`);
3. apply the factorial function to all integers in (`l`, `t`).

The same ideas are used for other generic functions. For instance, the module `GenEq` for the `StdGenerics` library offers the generic equality function `gEq`. Use this to

1. compute the equality of [1,2] and [1,2];
2. compute the equality of [1,2] and [2,3];
3. compute the equality of [1,2] and [2,3] where you use the less-then operator `<` for the elements of the list. This last item is only to show you the possibilities of the generic mechanism, it is not necessarily a recommended way of working.

Deadline

The deadline for this exercise is October 2 2017, 10:30h (just before the next lecture). Add the output of your programs as a comment.