Advanced Programming (I00032) 2019
Generics by overloading

Assignment 7

Goals of this assignment

In this exercise you program the transformation from datatypes to their generic representation and back to implement serialization in a generic way. This gives you a proper understanding of the generic mechanism.

1 Review Questions

Answer the following questions before implementing classes and instances in section 2. You can include the answers as a comment in your `.icl` file.

1. The definition of `==` for `UNIT` in the slides is

   ```
   instance == UNIT where (==) _ _ = TRUE
   ```

   This looks odd. Is it better to write

   ```
   instance == UNIT where
       (==) UNIT UNIT = TRUE
       (==) x    y    = FALSE
   ```

   or can we write

   ```
   instance == UNIT where (==) UNIT UNIT = TRUE
   ```

2. The definition for `(==)` for `CONS` is

   ```
   instance == (CONS a) | == a where (==) (CONS _ x) (CONS _ y) = x == y
   ```

   shouldn't we check the equality of constructor names as in

   ```
   instance == (CONS a) | == a where (==) (CONS a x) (CONS b y) = a == b && x == y
   ```

3. Given

   ```
   :: Bin   a =   Leaf | Bin (Bin a) a (Bin a)
   :: BinG  a :== EITHER (CONS UNIT) (CONS (PAIR (Bin a) (PAIR a (Bin a))))
   :: List  a =   Nil | Cons a (List a)
   :: ListG a :== EITHER (CONS UNIT) (CONS (PAIR a [a]))
   ```

   What are the generic representations of the values `[]` and `leaf`?

   Does this imply that `Leaf == []` yields `True` if we define it in the generic way?

2   Generic serialization

In assignment 1 we defined serialization of objects by a class

```
class serialize a where
  write :: a [String] → [String]
  read  :: [String] → Maybe (a,[String])
```

In this assignment we reimplement this class using the generic representation of data types:

```
:: UNIT       = UNIT
:: EITHER a b = LEFT a | RIGHT b
:: PAIR   a b = PAIR a b
:: CONS   a   = CONS String a
```

The types to be serialized are the native lists of Clean, and binary trees Bin. The generic representations ListG, and BinG are listed above. Define the transformation functions

```
fromList :: [a] → ListG a
toList   :: (ListG a) → [a]
fromBin  :: (Bin a) → BinG a
toBin    :: (BinG a) → Bin a
```

similar to transformations defined in the lecture slides.

2.1   With generic information

Define instances of serialize for [a] and Bin a based on their generic representation. Include all generic information in the serialized versions of data types, the serialized version contains strings like "UNIT" and "LEFT".

2.2   Without generic information

For humans it is nicer to omit the generic information. Only the basic types and the constructor names from the data types (like Leaf, Bin, Nil, and Cons) are included. For the write part this very simple. In the read it is at some places a little more advanced, especially in the instance for EITHER you need to backtrack. Be sure to include enough parenthesis whenever necessary.

Optional: Prettier serialization

Most likely your serialization of single constructor values like Leaf contains parenthesis, e.g. ["(","Leaf",")"] of the value Leaf. Beautify the serialization by omitting these parenthesis for constructors without arguments, e.g. ["Leaf"].

3   Reflection

When you implement everything correctly this will work fine for the listed types and all tests should pass. Can you come up with an example that breaks this system?

Although it is allowed to solve this problem, this is certainly not required.

Deadline

The deadline for this assignment is April 19 2021, 10:30h (just before the next lecture).

A   Getting started in Clean

1. Download the Clean system from your system:

   `https://ftp.cs.ru.nl/Clean/builds-quarterly/windows-x64/clean-bundle-complete-windows-x64-202104.zip`

   `https://ftp.cs.ru.nl/Clean/builds-quarterly/macos-x64/clean-bundle-complete-macos-x64-202104.tgz`

   `https://ftp.cs.ru.nl/Clean/builds-quarterly/linux-x64/clean-bundle-complete-linux-x64-202104.tgz`

2. Unzip the file to any convenient location. Follow the instructions in README.md

3. For Windows:

   (a) open the created folder

   (b) double click CleanIDE.exe

   (c) tell Windows this is fine

   (d) allow integration into the system

   (e) open a file `m.icl` and enter

   ```
   module m

   import StdEnv

   Start = 6 * 7
   ```

   the filename must match the module name

   (f) create a project from the file menu

   (g) press/select update and run from the Project menu

4. For Linux and MacOS

   create a file with the given contents using an editor of your choice

   ```
   module m

   import StdEnv

   Start = 6 * 7
   ```

   Follow the instructions on `https://clean.cs.ru.nl/*NIX_Instructions`.