Assignment 5
Mutual Recursion

15-814: Types and Programming Languages
Frank Pfenning & David M Kahn

Due Thursday, October 7, 2021
65 points

You should hand in a single file

- `hw05.cbv` with the code, where the solutions to the problems are clearly marked and auxiliary code (either from lecture or your own) is included so it passes the LAMBDA implementation, version v0.9. Auxiliary explanations should be included in the form of delimited comments `(* <comment> *)`.

Make sure to install the latest of LAMBDA (v0.9) or run it on `linux.andrew.cmu.edu`. You can find instructions on the software page on the course website, in the same place as before. Without this new version, you will not be able to test your code using `conv` in `*.cbv` files.

1 Type Isomorphisms

Task 1 (L9.1, 25 points) In `hw05.cbv`, implement the functions *ForthX* and *BackX* witnessing the each of the following isomorphisms *X* (where *X* ranges over i, ii, etc). You do not need to prove that they constitute an isomorphism, but you must test them to a certain extent. Details on the requirements for implementing and testing are given below.

(i) $0 \to \tau \cong 1$

(ii) $1 \to \tau \cong \tau$

(iii) $2 \to \tau \cong \tau \times \tau$

(iv) $\tau \times (\sigma + \rho) \cong (\tau \times \sigma) + (\tau \times \rho)$

(v) $(\sigma + \rho) \to \tau \cong (\sigma \to \tau) \times (\rho \to \tau)$

For this task, we ask that you restrict yourself to the pure language of Lecture 9 without recursion, where every function is terminating. Additionally, LAMBDA requires that all type variables are bound in `*.cbv` files. For example, in part (ii) you should have functions $\forall\alpha.\,(1 \to \alpha) \to \alpha$ and $\forall\alpha.\,\alpha \to (1 \to \alpha)$.

For the purpose of concrete testing, we recommend instantiating the type variables with $1 + 1$ (also known as *bool*) so you can observe the outcome of the test. Provide at least one test per per composition (so two per isomorphism) that demonstrate the composition behaves as the identity. These tests should use the `conv` declaration so that processing your file in lambda should fail if there is a counterexample. LAMBDA can't directly compare functions since their structure is not observable, so you have to provide some concrete inputs. We also ask that you group each test with the implementation it is testing, so that it can be easily identified when grading.

2 Programming with Lists

Task 2 (L10.3, 15 pts) Consider the type of lists of natural numbers

$$\mathit{list} = \mu\alpha.\, (\mathbf{nil} : 1) + (\mathbf{cons} : \mathit{nat} \times \alpha) \cong (\mathbf{nil} : 1) + (\mathbf{cons} : \mathit{nat} \times \mathit{list})$$

Define the following functions (including *plist*) in your `hw05.cbv` file. Feel free to use any definition of *nat* consistent with the natural numbers.

(i) $\mathit{nil} : \mathit{list}$, the empty list.

(ii) $\mathit{cons} : \mathit{nat} \times \mathit{list} \to \mathit{list}$, adding an element to a list. Include at least 1 test.

(iii) $\mathit{append} : \mathit{list} \to \mathit{list} \to \mathit{list}$, appending two lists. Include at least 1 test.

(iv) $\mathit{reverse} : \mathit{list} \to \mathit{list}$, reversing a list. Include at least 1 test.

(v) $\mathit{itlist} : \mathit{list} \to \forall\beta.\, (\mathit{nat} \times \beta \to \beta) \to \beta \to \beta$ satisfying

$$\begin{array}{lcl} \mathit{itlist}\ \mathit{nil}\ [\tau]\ f\ c & = & c \\ \mathit{itlist}\ (\mathit{cons}\ \langle n, l\rangle)\ [\tau]\ f\ c & = & f\ \langle n, \mathit{itlist}\ l\ [\tau]\ f\ c\rangle \end{array}$$

where you may take equality to be extensional. This captures *iteration* over lists, for the special case where the elements are all natural numbers. You do not need to prove the correctness of your representation, nor provide any testing.

(vi) Design a type and implementation for *primitive recursion* over lists, defining a function *plist*. Note that we do *not* ask for primitive recursion over the naturals contained in the list, only over the list itself. You do not need to prove the correctness of *plist*, nor provide any testing.

3 Mutually Recursive Types

Task 3 (L10.4, 25 points) It is often intuitive and useful to define types in a mutually recursive way. For example, we might specify the even and odd natural numbers in unary representation with the following desired isomorphisms:

$$\begin{array}{lcl} \mathit{even} & \cong & (\mathbf{zero} : 1) + (\mathbf{succ} : \mathit{odd}) \\ \mathit{odd} & \cong & () + (\mathbf{succ} : \mathit{even}) \end{array}$$

Here the empty parenthesis $()$ are used to indicate that $(\mathbf{succ} : \mathit{even})$ is a disjoint sum with just a single alternative. The only value v of type *odd* would be $\mathsf{fold}\ (\mathbf{succ} \cdot v')$ with $v' : \mathit{even}$. Part of this task will be to find a representation of such types using the explicit recursive type constructor $\mu\alpha.\, \tau$.

Let the type of bit strings (which, during lecture, we used to represent numbers in binary form) be defined as

$$\begin{array}{rcl} \mathit{bits} & \cong & (\mathbf{b0} : \mathit{bits}) + (\mathbf{b1} : \mathit{bits}) + (\mathbf{e} : 1) \\ \mathit{bits} & = & \mu\alpha.\, (\mathbf{b0} : \alpha) + (\mathbf{b1} : \alpha) + (\mathbf{e} : 1) \end{array}$$

We say a bit string has parity 0 if it has an even number of 0s and 1 if it has an odd number of 1s. The answer to the questions below should be included in the file `hw05.cbv`.

(i) Define isomorphisms to be satisfied by two types *bits0* and *bits1*, where the values of type *bits0* are exactly the bit strings with parity 0, and the values of type *bits1* are exactly the bit strings with parity 1.

(ii) Give explicit definitions $\mathit{bits0} = \ldots$ and $\mathit{bits1} = \ldots$ using the recursive type constructor $\mu\alpha.\,\tau$ satisfying this specification.

(iii) We now define a type

$$\mathit{parity} \quad = \quad (\mathbf{p0} : 1) + (\mathbf{p1} : 1)$$

Define a function $\mathit{parity} : \mathit{bits} \to \mathit{parity}$ that computes the parity of the given bit string.

(iv) Next we define

$$\begin{array}{rcl} \mathit{par0} & = & (\mathbf{p0} : 1) + () \\ \mathit{par1} & = & () + (\mathbf{p1} : 1) \end{array}$$

It should be the case that

$$\begin{array}{llll} \mathit{parity}\ v_0 & \mapsto^* & w_0 & \text{where } w_0 : \mathit{par0} \text{ if } v_0 : \mathit{bits0} \\ \mathit{parity}\ v_1 & \mapsto^* & w_1 & \text{where } w_1 : \mathit{par1} \text{ if } v_1 : \mathit{bits1} \end{array}$$

Does your implementation of *parity* have either following types?

$$\begin{array}{rcl} \mathit{parity} & : & \mathit{bits0} \to \mathit{par0} \\ \mathit{parity} & : & \mathit{bits1} \to \mathit{par1} \end{array}$$

If not, explain why not. We are not looking for a paraphrase of the error message, but a brief analysis why the two types above may be difficult to verify for a type-checker.

If yes, explain briefly which feature of your implementation made it possible for the type-checker to verify both of these properties.

The explanations should be included your `hw05.cbv` file. You may use the delimited comments `(* <comment> *)` for this purpose.