Exercises on realizability
(Midlands Graduate School 2022)

Andrej Bauer

April 10, 2022

Instructions

These are the exercises accompanying the "Realizability" lectures, given at the Midlands Graduate School 2022. During the exercise classes we shall solve them in the following way:

1. I will explain the exercise and review the background knowledge.
2. You will talk to your neighbor for 10 minutes about the exercise.
3. We will discuss the solution together.

You are of course welcome to work on the exercises ahead of the class. Please consult the accompanying "Notes on realizability" (PDF, GitHub repository) and discuss the exercises on the MGS Discord server.

1 Lecture 1: Models of computation

1.1 Dove-tailing

Let φ_n be the partial computable function computed by the Turing machine encoded by $n \in \mathbb{N}$. Write $e{\downarrow}$ to indicate that the value of e is defined. Show that there is a partial computable function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ such that, for all $n, k \in \mathbb{N}$,

$$fn = k \Rightarrow \varphi_n(k){\downarrow} \quad \text{and} \quad (\exists m \in \mathbb{N}\,.\, \varphi_n(m){\downarrow}) \Rightarrow fn{\downarrow}.$$

You are not expected to construct an actual Turing machine, but you should describe in some detail how the machine works.

Is there a Haskell function `f : (Int → Int) → Int` with an analogous property?

1.2 Modulus of continuity

Write $\ulcorner e \urcorner$ for the encoding of entity e as a natural number, and let $\mathbb{B} = \mathbb{N}^{\mathbb{N}}$ be the Baire space. Given $\alpha \in \mathbb{B}$, let $\overline{\alpha}(k) = [\alpha 0, \ldots, \alpha(k-1)]$. Write $0^{(\omega)}$ for the sequence of all zeroes and $0^{(k)}$ for the list of k zeroes $[0, \ldots, 0]$.

Recall how $\gamma \in \mathbb{B}$ encodes a map $\boldsymbol{\eta}_\gamma : \mathbb{B} \rightharpoonup \mathbb{B}$. Given input $\alpha \in \mathbb{B}$ and $i \in \mathbb{N}$, we compute the value of $\boldsymbol{\eta}_\gamma\, \alpha\, i$ by looking up in succession

$$\begin{aligned}
&\gamma(\ulcorner i{::}[]\urcorner),\\
&\gamma(\ulcorner i{::}[\alpha 0]\urcorner),\\
&\gamma(\ulcorner i{::}[\alpha 0, \alpha 1]\urcorner),\\
&\gamma(\ulcorner i{::}[\alpha 0, \alpha 1]\urcorner),\\
&\gamma(\ulcorner i{::}[\alpha 0, \alpha 1, \alpha 2]\urcorner),\\
&\vdots
\end{aligned}$$

until we find the first non-zero one $j > 0$ and output $j - 1$. More precisely, define $\ell(\gamma, \alpha) : \mathbb{N} \rightharpoonup \mathbb{N}$ by

$$\ell(\gamma, \alpha)(i) = \gamma(\ulcorner i{::}\overline{\alpha}(k)\urcorner) - 1 \quad \text{where} \quad k = \min_k(\gamma(\ulcorner i{::}\overline{\alpha}(k)\urcorner) \neq 0)$$

(if not such k exists then $\ell(\gamma, \alpha)(i)$ is undefined), and let the map $\boldsymbol{\eta}_\gamma : \mathbb{B} \rightharpoonup \mathbb{B}$ encoded by γ be

$$\boldsymbol{\eta}_\gamma(\alpha) = \begin{cases} \ell(\gamma, \alpha) & \text{if } \ell(\gamma, \alpha) \text{ is a total map,} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Suppose $f : \mathbb{B} \to \mathbb{B}$ is a *total* map satisfying

$$\forall k \in \mathbb{N}.\, \exists m \in \mathbb{N}.\, \forall \alpha \in \mathbb{B}.\, \overline{\alpha}(m) = 0^{(k)} \Rightarrow (f\alpha)k = (f0^{(\omega)})k. \tag{1}$$

We say that m is a *modulus of continuity* for f at k.

Verify that f satisfies (1) if, and only if, it is continuous with respect to the product topology on $\mathbb{B}$.

Construct a Type 2 Turing machine which accepts as input $\gamma \in \mathbb{B}$ and $k \in \mathbb{N}$ and outputs a modulus of continuity for $\boldsymbol{\eta}_\gamma$ at k. We assume that γ encodes a *total* map $\mathbb{B} \to \mathbb{B}$. Concretely, the read-only input tape contains k followed by γ (in Type 2 computability tape cells contain numbers or blanks). The machine should terminate and output a suitable m. Of course, it suffices to describe the machine informally.

Is there a corresponding Haskell map `modulus` : `Int` $\to$ (`Int` $\to$ `Int`) $\to$ `Int`?

1.3 Programming in λ-calculus

Use the online untyped λ-calculus interpreter at

`http://www.andrej.com/zapiski/MGS-2022/lambda/`

to implement a function which tests whether a given natural number is prime. Consult the notes section on the λ-calculus to get basic arithmetic and Booleans going. If you prefer to use the much faster off-line interpreter, you can get compile the language `lambda` from the Programming Languages Zoo – but refrain from looking at examples because they contain the solution.

The actual programming should be done in the privacy of your computer. During the exercise class you should just write down a list of specific functions (zero testing, searching, division, etc.) that you need to implement. Assuming you have implemented those, write down the primality testing function. To get you started, consult Figure 1 (where ^ stands for λ):

```
-- the constant function
K := ^ x y . x ;

-- pairing
pair := ^ a b . ^p . p a b ;
first := ^ p . p (^x y . x) ;
second := ^ p . p (^x y. y) ;

-- Booleans
true  := ^x y . x ;
false := ^x y . y ;
if := ^u . u ;

-- recursive definitions
fix := ^f . (^x . f (x x)) (^x . f (x x)) ;

-- arithmetic
0  := ^f x . x ;
1  := ^f x . f x ;
2  := ^f x . f (f x) ;
3  := ^f x . f (f (f x)) ;
+ := ^n m f x . (n f) ((m f) x) ;
* := ^n m f x . (n (m f)) x ;
succ := ^n f x . f (n f x) ;
pred := ^n . second (n (^p. pair (succ (first p)) (first p)) (pair 0 0)) ;
iszero := ^n . (n (K false)) true ;
```

Figure 1: Basic programming in `lambda`.