Math. Struct. in Comp. Science (2015), *vol.* 25, *pp.* 1484–1489.
doi:10.1017/S0960129513000406 First published online 10 November 2014

An injection from the Baire space to natural numbers

ANDREJ BAUER

Faculty of Mathematics and Physics, University of Ljubljana, Ljubljana, Slovenia
Email: andrej.bauer@andrej.com

Received 13 August 2012

We provide a realizability model based on infinite time Turing machines in which there is an injection from the internal Baire space, the object of infinite sequences of numbers, to the object of natural numbers.

1. Introduction

At the Mathematical Foundations of Programming Semantics XXVII in May 2011, Oliva (2011) held a tutorial in which he showed a program witnessing the fact that there was no injection from the Baire space $\mathbb{N}^{\mathbb{N}}$ to natural numbers $\mathbb{N}$. The program took as input a function $h : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ and produced two sequences $x, y \in \mathbb{N}^{\mathbb{N}}$ such that $x \neq y$ and $h(x) = h(y)$. Martín Escardó popularized the program as an interesting example of extraction of computational content from classical proofs, which left one wondering whether there was a constructive proof of the statement

$$\forall h : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}. \exists x, y \in \mathbb{N}^{\mathbb{N}}. x \neq y \wedge h(x) = h(y) \tag{1}$$

that would yield such a program more directly. Fred Richman asked for a constructive proof of the weaker statement that there was no injection from the Baire space to the natural numbers, and nobody could come up with one.

Classically there is no injection $h : \mathbb{N}^{\mathbb{N}} \to \mathbb{N}$, of course. Constructively, such a map must be wildly discontinuous, if it exists. Indeed, because convergent sequences in $\mathbb{N}$ are eventually constant, h must map *every* injective sequence to a non-convergent sequence in $\mathbb{N}$. As every point is the limit of an injective sequence, h must be discontinuous at every point. Brouwerian intuitionism, Russian constructivism and many other familiar models of constructive mathematics, all enjoy continuity principles which therefore prohibit such discontinuous maps.

Let me also mention that in Russian constructivism and in the effective topos (Hyland 1982), an injection $\mathbb{N}^{\mathbb{N}} \to \mathbb{N}$ 'almost' exists because the Baire space $\mathbb{N}^{\mathbb{N}}$ is a quotient of a subset of $\mathbb{N}$. This gives us a *multi-valued* injective map $\mathbb{N}^{\mathbb{N}} \to \mathbb{N}$, which however cannot be made single valued. Or to put it in another way, we have an injective operation which does not preserve extensional equality of functions.

In this note, I observe that there is a realizability topos based on infinite time Turing machines (Hamkins and Lewis 2000) in which there is an injection $\mathbb{N}^{\mathbb{N}} \to \mathbb{N}$. Consequently, we cannot hope to extract Oliva's program directly from a constructive proof (1). It is

likely that the topos can be used for other ominous purposes. For example, it validates the principle LPO, in fact it decides any arithmetical statement, but its logic is not classical, as it never is in a non-trivial realizability topos.

2. Infinite time Turing machines

We give a brief overview of infinite time Turing machines and recommend Hamkins and Lewis (2000) as a reference which contains more detailed descriptions and proofs of all the unsupported claims made here. An infinite time Turing machine, or just *machine*, is like a Turing machine which is allowed to run infinitely long, where the computation steps are counted by ordinals. The machine has a finite program, an input tape, work tapes, an output tape, etc. We assume that the tape cells contain 0's and 1's. At successor ordinals, the machine acts like an ordinary Turing machine. At limit ordinals it enters a special 'limit' state, its heads are placed at the beginnings of the tapes, and the content of each tape cell is computed as the lim sup of the values written in the cell at earlier stages. More precisely, if c_α denotes the value of the cell c at step α, then for a limit ordinal β we have

$$c_\beta = \begin{cases} 0 & \text{if } \exists \alpha < \beta \,.\, \forall \gamma \,.\, (\alpha \leqslant \gamma < \beta \Rightarrow c_\alpha = 0), \\ 1 & \text{otherwise.} \end{cases}$$

The machine terminates by entering a special halt state, or it may run forever. It turns out that a machine which has not terminated by step ω_1 runs forever.

We can think of machines as computing partial functions $2^{\mathbb{N}} \rightharpoonup 2^{\mathbb{N}}$: we initialize the input tape with an infinite binary sequence $x \in 2^{\mathbb{N}}$, run the machine, and observe the contents of the output tape if and when the machine terminates. We can also consider infinite time computation of partial functions $\mathbb{N} \rightharpoonup \mathbb{N}$: we initialize the input tape with the input number, run the machine and interpret the contents of the output tape as a natural number, where we ignore anything that is beyond the position of the output head. By performing the usual encoding tricks, we can feed the machines more complicated inputs and outputs, such as pairs, finite lists and even infinite lists of numbers. We say that a function is *infinite time computable* if there is a machine that computes it.

The power of infinite time Turing machines is vast and extends far beyond the halting problem for ordinary Turing machines, although, of course they cannot solve their own halting problem. For example, for every Π^1_1-subset $S \subseteq 2^{\mathbb{N}}$ there is a machine which, given $x \in 2^{\mathbb{N}}$ on its input tape, terminates and decides whether $x \in S$.

There is a standard enumeration $t_0, t_1, t_2, \ldots$ of infinite time Turing machines, where t_n is the machine whose program is encoded by the number n in some reasonable manner. The associated enumeration $\psi_0, \psi_1, \psi_2, \ldots$ of infinite time computable partial functions $\mathbb{N} \rightharpoonup \mathbb{N}$ is defined as

$$\psi_n(k) = \begin{cases} m & \text{if } t_n \text{ on input } k \text{ terminates and outputs } m, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The enumeration ψ satisfies the s-m-n and u-t-m theorems.

https://doi.org/10.1017/S0960129513000406 Published online by Cambridge University Press

Theorem 2.1 (s-m-n). There is a total infinite time computable map $s : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that $\psi_{s(m,i)}(j) = \psi_m(\langle i, j\rangle)$ for all $m, i, j \in \mathbb{N}$.

Theorem 2.2 (u-t-m). There is a partial infinite time computable map $u : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathbb{N}$ such that $\psi_n(k) = u(n, k)$ for all $n, k \in \mathbb{N}$.

To convince ourselves that the u-t-m theorem holds, we think a bit how a universal infinite time Turing machine works. It accepts the description n of a machine and the initial input tape x. At successor steps, the simulation of machine t_n on input x proceeds much like it does for the ordinary Turing machines. Thus it takes finitely many successor steps to simulate one successor step of t_n. Each limit step of t_n is simulated by one limit step of the universal machine, followed by finitely many successor steps. Indeed, whenever the universal machine finds itself in the special limit state, it puts the simulated machine in the simulated limit state, and moves the simulated heads to the beginnings of the simulated tapes. These actions take finitely many steps. The contents of the simulated tapes need not be worried about, as it will be updated correctly at limit stages.

To see what sort of tasks can be performed by infinite time Turing machines, we consider several examples that will be useful later on.

There is a machine which decides whether two infinite sequences $x, y \in \mathbb{N}^{\mathbb{N}}$ are equal. It first initializes a fresh work cell with 0, and then for each k, it compares x_k and y_k. If they differ, it sets the work cell to 1. After ω steps the work cell will be 1 if and only if, $x \neq y$.

A more complicated problem is to *semi*decide whether a given machine t_n computes a given sequence $x \in 2^{\mathbb{N}}$. The machine which performs such a task accepts n and x as inputs and begins by writing down the sequence $y_k = \psi_n(k)$ onto a work tape. This it can do by simulating t_n successively on inputs $0, 1, 2, \ldots$ and writing down the values y_k as they are obtained. The machine also keeps track of which values y_k have been computed by flipping bits on a separate 'tally' tape from 0 to 1. If any of the y_k's is undefined, the machine will run forever. Otherwise, it will be able to detect in ω steps that the entire sequence y has been computed and written down by checking that all bits on the separate 'tally' tape have been flipped to 1. After that, the machine verifies that $x_k = y_k$ for all $k \in \mathbb{N}$, as described previously.

Suppose, we have a machine t which expects as input an infinite sequence x and a number n. We would like to construct another machine which accepts an infinite sequence x and outputs a number n such that $t(x, n)$ terminates, if one exists. We use the familiar dovetailing technique to tackle the problem. Given $x \in 2^{\mathbb{N}}$ as input, we simulate in parallel the executions of machine t on inputs of the form (x, n), one for each n:

$$t(x, 0), \quad t(x, 1), \quad t(x, 2), \quad \ldots$$

Each of these requires several infinite tapes, but since we only need countably many of them, they may be interleaved into a single tape. At successor steps, the simulation performs the usual dovetailing technique. At limit steps, the simulation inserts extra ω bookkeeping steps, during which it places the simulated machines in the 'limit' state and moves their head positions. The extra steps do not ruin the limits of the simulated tapes, because those are left untouched. After the extra steps are performed, dovetailing starts

https://doi.org/10.1017/S0960129513000406 Published online by Cambridge University Press

over again. As soon as one of the simulations $t(x,n)$ terminates, we return the results n. Note that n is computed from x in a deterministic fashion, although a different simulation technique may yield a different n.

3. Realizability over infinite time Turing machines

For background on realizability theory, we refer to van Oosten (2008). To build a realizability model from infinite time Turing machines, we first need to turn them into a partial combinatory algebra $\mathbb{J}$. Because the infinite time Turing machines enjoy the s-m-n and u-t-m theorems, this is no problem at all. The underlying set of $\mathbb{J}$ is the set of natural numbers $\mathbb{N}$, and the partial application operation applies m to n by computing $\psi_m(n)$. The combinator K is obtained by an application of the s-m-n theorem to the first projection $\langle i,j\rangle \mapsto i$, while the combinator S requires a bit more work and the use of the u-t-m theorem.

In the next section, we will show that the realizability topos $\mathsf{RT}(\mathbb{J})$ contains an injection $\mathsf{N}^\mathsf{N} \to \mathsf{N}$. In fact, we only need to consider a much simpler realizability model of *numbered sets* over $\mathbb{J}$. These are equivalent to a full subcategory of $\mathsf{RT}(\mathbb{J})$ which contains the natural numbers object N and the internal Baire space N^N.

Recall that a numbered set (S,δ) is a set S with a partial surjection $\delta : \mathbb{N} \rightharpoonup S$. If (T,η) is another numbered set, we say that a map $f : S \to T$ is infinite time computable with respect to δ and η if there exists an infinite time computable map $r : \mathbb{N} \rightharpoonup \mathbb{N}$ such that $\mathrm{dom}(\delta) \subseteq \mathrm{dom}(r)$ and $f(\delta(n)) = \eta(r(n))$ for all $n \in \mathrm{dom}(\delta)$. We say that r *tracks* f.

The natural numbers object in $\mathsf{RT}(\mathbb{J})$ is the numbered set $\mathsf{N} = (\mathbb{N}, \mathrm{id}_\mathbb{N})$, while the internal exponential N^N is the set

$$\{f : \mathbb{N} \to \mathbb{N} \mid f \text{ is infinite-time computable}\}$$

of infinite time computable total function, with the numbering ψ restricted to the codes of total maps.

4. An injection $\mathsf{N}^\mathsf{N} \to \mathsf{N}$ in $\mathsf{RT}(\mathbb{J})$

To show that our realizability model has in injection $\mathsf{N}^\mathsf{N} \to \mathsf{N}$, we first formulate a constructive plan of attack. Recall that a set is *subcountable* if it is the image of a subset of $\mathbb{N}$.

Proposition 4.1. Suppose the following hold:

1. choice from functions to numbers, and
2. $\mathbb{N}^\mathbb{N}$ is a subcountable set.

Then there is an injection $\mathbb{N}^\mathbb{N} \to \mathbb{N}$, constructively.

Proof. The second condition means that there is a partial surjection $e : \mathbb{N} \rightharpoonup \mathbb{N}^\mathbb{N}$. For all $f \in \mathbb{N}^\mathbb{N}$, there exists $n \in \mathbb{N}$ such that $e(n) = f$. By function choice there exists $h : \mathbb{N}^\mathbb{N} \to \mathbb{N}$ such that $e(h(f)) = f$ for all $f \in \mathbb{N}^\mathbb{N}$. Clearly, h is injective because e is its right inverse. □

https://doi.org/10.1017/S0960129513000406 Published online by Cambridge University Press

The second condition is satisfied in $\mathsf{RT}(\mathbb{J})$. The partial surjection $e : \mathsf{N} \rightharpoonup \mathsf{N}^{\mathsf{N}}$ is defined as

$$e(n) = \begin{cases} \psi_n & \text{if } \psi_n \text{ is total} \\ \text{undefined} & \text{otherwise,} \end{cases}$$

and is tracked by the identity map. Its domain of definition is the set of codes of total infinite time computable maps. The map e is surjective in the internal logic of the topos because it is surjective and tracked by the identity.

Choice from functions to numbers is also known as $AC_{1,0}$. It states that any total relation between $\mathbb{N}^{\mathbb{N}}$ and $\mathbb{N}$ contains a function. We show that $\mathsf{RT}(\mathbb{J})$ satisfies an even stronger principle, namely general choice for functions: every total relation on $\mathbb{N}^{\mathbb{N}}$ contains a function. In categorical terms, this amounts to N^{N} being internally projective, see van Oosten (2008, 3.2.3).

Proposition 4.2. The object N^{N} is internally projective in $\mathsf{RT}(\mathbb{J})$, if and only if, there exists an infinite time computable map $r : \mathbb{N} \rightharpoonup \mathbb{N}$ such that:

1 if ψ_k is total then $r(k)$ is defined and $\psi_k = \psi_{r(k)}$, and
2 if ψ_k is total then $r(r(k)) = r(k)$.

Proof. See e.g. van Oosten (2008, 3.2.3) or Bauer (2000, 1.3.4). □

We describe informally how a machine t computing r works. Suppose k is the code of a total function ψ_k (our machine will diverge if k is not the code of a total function). The machine t first writes down the sequence $x_i = \psi_k(i)$ onto a tape. At the end of Section 2, we argued that there is a machine which accepts x and a number m, and terminates if and only if, ψ_m computes x. Therefore, by dovetailing t can compute an m such that ψ_m computes x. Crucially, m depends only on x and the particular dovetailing technique, but not on k. We may take $r(k) = m$ because $\psi_k = x = \psi_m$, and $r(r(k)) = m$.

The map r just described tracks an injection $\mathsf{N}^{\mathsf{N}} \to \mathsf{N}$, so we could have constructed one directly, without knowing anything about internally projective sets. Nevertheless, it is still interesting to know that $\mathsf{RT}(\mathbb{J})$ validates function choice.

Acknowledgements

I thank Joel Hamkins for explaining infinite time Turing machines to me, and to Alex Simpson and Jaap van Oosten for helpful discussions.

References

Bauer, A. (2000) *The Realizability Approach to Computable Analysis and Topology*, Ph.D. thesis, Carnegie Mellon University.

Hamkins, J. D. and Lewis, A. (2000) Infinite time turing machines. *Journal of Symbolic Logic* **65** (2) 567–604.

Hyland, J. (1982) The effective topos. In: Troelstra, A. and Dalen, D. V. (eds.) *The L.E.J. Brouwer Centenary Symposium*, North Holland Publishing Company 165–216.

Oliva, P. (2011) Programs from classical proofs via Gödel's dialectica interpretation. In: *27th Conference on Mathematical Foundations of Programming Semantics (MFPS XXVII)*, Pittsburgh, USA.

van Oosten, J. (2008) *Realizability: An Introduction to its Categorical Side*, Studies in Logic and the Foundations of Mathematics volume 152, Elsevier.