1. The proof of the no-go theorem (Corollary 3)

While the no-go theorem (Corollary 3) is itself correct, the intermediate result used to justify it (Theorem 2) is not. We are grateful to Sean Moss for pointing out this oversight. We present a revised proof of Corollary 3 which parallels the techniques presented in Berger and Setzer [BS18].

We recall the statement of the no-go theorem (Corollary 3):

**Theorem 1.1.** Conversion is undecidable in a guarded type theory satisfying the following requirements:

- there is a type $S$ equipped with a definitional isomorphism $S \cong \text{nat} \times \top S$.
- next : $A \to \top A$ is injective on closed terms,
- the lob operator unfolds.

We fix some effective encoding of Turing machines $\text{TM} = \text{nat}$ as well as an encoding of the state of the tape of the machine as another natural number $\text{State} = \text{nat}$. We further assume the following operations are defined:

1. $\text{init} : \text{TM} \to \text{State}$
2. $\text{step} : \text{TM} \times \text{State} \to \text{State}$
3. $\text{halt} : \text{TM} \times \text{State} \to \text{bool}$
4. $\text{result} : \text{State} \to \text{nat}$

These operations parallel those required by Berger and Setzer [BS18] and are well-known to be definable in type theory—in fact, they are primitive recursive. Using these operations, we can define an element of $S$ which represents the (potentially non-terminating) trace of a machine:

exec : $\text{TM} \to S$
exec$(x) = \text{go}(\text{init}(x))$

go : $\text{State} \to S$
go = $\text{lob}(r. \lambda s. \text{if} \ \text{halt}(x, s) \ \text{then} \ \text{const}(\text{result}(s)) \ \text{else} \ \text{cons}(0, r \ \otimes \ \text{next}(\text{step}(x, s))))$

Assuming that a machine $x$ terminates in $n$ steps with result $\epsilon$, and using the fact that lob unfolds, we therefore conclude the following:

exec$(x) = \text{cons}(0, \ldots, \text{next}(\text{cons}(0, \text{next}(\text{const}(\epsilon))))))$

Accordingly, using the $\eta$ law for dependent sums and the definitional isomorphism for $S$ we see that if $x$ terminates with result 0 then exec$(x) = \text{const}(0)$. To see this, suppose $x$ terminates in 2 steps with result 0:

exec$(x)$

---

*Date:* Wednesday 31st August, 2022.
\[= \text{go}(s_0)\]  
\[s_0 = \text{init}(x)\]

\[= \text{cons}(0, \text{next} (\text{go}(s_1)))\]  
\[s_1 = \text{step}(x, s_0)\]

\[= \text{cons}(0, \text{next} (\text{cons}(0, \text{next} (\text{const}(0))))))\]

\[= \text{cons}(0, \text{next} (\text{const}(0)))\]

Using the \(\eta\) law and definition of \text{const}

\[= \text{const}(0)\]

This argument is easily seen to extend to programs taking an arbitrary but finite number of steps to terminate with result 0. Furthermore, if \(x\) terminates in \(n\) steps with result 1 then \(\text{exec}(x)\) is not definitionally equal to \(\text{const}(0)\); the former has \(\text{next}^n(1)\) for the \(n\)th element while the latter has \(\text{next}^n(0)\) and \(\text{next}\) is assumed to be injective on closed terms.

To complete our proof, we require the following classical result:

**Theorem 1.2** (Rosser [Ros36], Kleene [Kle50], Trahtenbrot [Tra53]). Consider the following subsets of natural numbers:

\[A = \{x \mid x \text{ is a Turing machine terminating with result 0}\}\]

\[B = \{x \mid x \text{ is a Turing machine terminating with result 1}\}\]

\(A\) and \(B\) are recursively inseparable. In other words, there is no computable function \(\mathbb{N} \to \mathbb{N}\) which sends \(A\) to 0 and \(B\) to 1 while terminating on all inputs.

We are now in a position to prove the no-go theorem:

**Proof of Theorem 1.1.** Suppose that conversion was decidable. Precisely, fix an effective encoding of terms of guarded type theory as natural numbers and assume there is a total computable function \(e : \mathbb{N} \times \mathbb{N} \to \mathbb{B}\) which decides conversion.

In this case, consider the following computable function:

\[\lambda n. \text{e}(\text{"exec(suc"}(\text{zero}))\","\text{const}(0)\")\]

As \(e\) is assumed to be total, this is a total computable function. However, we have just seen that this function separates Turing machines halting with 0 and those halting with 1, contradicting Theorem 1.2. \(\square\)

**References**


---

1We have followed the attribution used by Berger and Setzer [BS18] for this result. There the authors also point to textbook reproductions of this fact.