Applications of the generalized Cantor theorem
Anwendungen des verallgemeinerten Satzes von Cantor

Tex Schönlank

2024-04-13

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

1 Introduction

In 1897, Georg Cantor provided a proof that the set $\mathbb{N}$ of natural numbers is in some sense strictly smaller than the powerset $2^{\mathbb{N}}$ of $\mathbb{N}$, i.e. the set of all subsets of $\mathbb{N}$. The technique they used is now commonly known as a *diagonal argument*.

The diagonal technique appears many times in mathematical logic. In 1969, F. William Lawvere noted that Russell's paradox (the barber paradox), Gödel's incompleteness theorem and Tarski's undefinability theorem all seem to share their diagonal character.[Law69] They use the setting of cartesian-closed categories to derive a generalized version of the diagonal argument. We call this the *Generalized Cantor theorem* (GCT).

In 2003, Noson S. Yanofsky published a paper in which they reproduce this idea in a non-categorical setting.[Yan03] They apply the GCT to many more results from mathematics.

In our paper, we investigate one suggestion and one claim from [Yan03]. The outline of this document is as follows.

In Section 2 we introduce the necessary background material. We start with the necessary minimum of category theory, explain Cantor's original argument and how it can be rephrased in categorical language.

Then, in Section 3, we dive into computability theory. We introduce the notions of recursivity and primitive recursivity and a specific, classic function from computability theory, the Ackermann function. It is known that the Ackermann function is recursive but not primitive recursive. We investigate Yanofsky's suggestion that this can be proven using the GCT. Ultimately we conclude that it does not seem to be possible, at least using approaches similar to ours.

Finally, in Section 4, we look at another result from computability theory, the recursion theorem. Yanofsky's paper included a putative proof of this theorem using the GCT. In this document, we scrutinize Yanofsky's proof and reveal a few flaws. We attempt to iron out the kinks in the spirit of the GCT. It turns out, however, that our various steelmanned versions of the proof, which all necessarily reformulate the statement, also fail.

2 Background

We begin with a remark on notation, some of which is borrowed from [Cut80].

$A \to B$ denotes the set of all total functions from A to B. $A \rightharpoonup B$ denotes the set of all partial functions from A to B.

Most of the time, 0 is a natural number. In the cases where it is not, it will either be clear or not matter terribly much.

We use the equivalence symbol $\simeq$ as in [Cut80]. Suppose e and e' are mathematical expressions (one might say terms). When either of them contains the application of partial functions it can happen that that expression as a whole is undefined. The statement "$e \simeq e'$" is the extension of "$e = e'$" to this case. $e \simeq e'$ means: e is defined iff e' is defined and when both are defined then they are equal.

Definition 1. *A set X is* denumerable *iff it is in bijection with* $\mathbb{N}$.

A set X is enumerable *iff there is a surjection from* $\mathbb{N}$ *to X. This surjection is called an* enumeration of X.

2.1 Category theory

We start this section with a very quick introduction to the basic definitions of category theory. We will not be bothered by the foundations and base ourselves on the usual, vague concept of 'collection'.

A *category* $\mathbf{C}$ consists of: a collection $\mathrm{ob}\,\mathbf{C}$, the elements of which are called the *objects*; for every two objects A, B of $\mathbf{C}$ the collection $\mathrm{Hom}_{\mathbf{C}}(A, B)$, the elements of which are called the *morphisms (or arrows) from A to B* (we often write $\mathrm{Hom}(A, B)$ if the category is clear and we write $f : A \to B$ if f is in $\mathrm{Hom}(A, B)$); a notion of equality of objects as well as one of morphisms, both of which behave as equality does in first-order logic, i.e. they are reflexive, symmetric, transitive and congruent; a *composition operator* $\circ$ that maps all f in $\mathrm{Hom}(A, B)$ and g in $\mathrm{Hom}(B, C)$ to $g \circ f$ in $\mathrm{Hom}(A, C)$ and which is associative (i.e. $(h \circ g) \circ f = h \circ (g \circ f)$ whenever these are defined); and for every object A a morphism Id_A (the *identity morphism of* A) in $\mathrm{Hom}(A, A)$ such that $g \circ \mathrm{Id}_A = \mathrm{Id}_A = \mathrm{Id}_A \circ f$ whenever these are defined.

Note that there is no more than one identity morphism per object. If $\mathrm{Id}_A, \mathrm{Id}'_A$ are both identity morphisms of A then $\mathrm{Id}_A = \mathrm{Id}_A \circ \mathrm{Id}'_A = \mathrm{Id}'_A$.

A morphism $f : A \to B$ is an *isomorphism* iff there exists a morphism $g : B \to A$ such that $g \circ f = \mathrm{Id}_A$ and $f \circ g = \mathrm{Id}_B$. Objects A, B between which an isomorphism exists are called *isomorphic*. In category theory we rarely care about specific objects; isomorphic objects often suffice as they share any properties in which we are interested.

The equality of morphisms plays an important role in category theory. Aside from formulas we also use diagrams to express this. Consider the diagrams below. If $h \circ f = i \circ g$ we say that *the square $ABCD$ commutes.* If $k \circ j = l$ we say that *the triangle EFG commutes* and that l *factors through j.*

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g"'] & B \arrow[d, "h"] \\
C \arrow[r, "i"'] & D
\end{tikzcd}
\qquad
\begin{tikzcd}
E \arrow[r, "j"] \arrow[dr, "l"'] & F \arrow[d, "k"] \\
& G
\end{tikzcd}$$

Another important concept in category theory is *diagram chasing.* Consider the diagrams below:

$$\begin{tikzcd}
A \arrow[r] \arrow[d] \arrow[dr] & B \arrow[d] & G \arrow[d] \arrow[ddr] & \\
C \arrow[r] \arrow[d] \arrow[dr] & D \arrow[d] & H \arrow[d] \arrow[dr] & \\
E \arrow[r] & F & I \arrow[r] & J
\end{tikzcd}$$

Suppose that all unintersected triangles (that is, ABD, ACD, CDF, CEF, GHJ and HIJ) are commuting triangles. We can combine the commuting triangles ABD and ACD to conclude that the square $ABCD$ commutes as well, i.e. the morphisms ABD and ACD are equal. In a similar but slightly different way we see that ACD and CDF combine to yield the square commutativity of ADF with ACF. By repeating such combinations we obtain the commutativity of all compositions of morphisms in the shown diagrams. A note on terminology: commutativity of ABD with ACD is called the commutativity of an *inner square* of the diagram. The commutativity of $ABDF$ with $ACEF$ is referred to as that of the *outer square*. Similarly the commutativity of GHJ with GJ is that of an *inner triangle* and the commutativity of $GHIJ$ with GJ is that of the *outer triangle*. This is purely due to the chosen visual representation.

An object P in a category $\mathbf{C}$ is called a *terminal object* iff, for every object Z, there is exactly one morphism from Z to P. Note that all terminal objects are isomorphic: if P, P' are terminal then there exists exactly one $f : P \to P'$ and one $g : P' \to P$, all morphisms $P \to P$ are equal to Id_P and similarly for P'. Then $g \circ f = \mathrm{Id}_P$ and $f \circ g = \mathrm{Id}_{P'}$, which means f, g are isomorphisms. Since terminal objects are isomorphic we refer to *the* terminal object as 1 and do not care which specific object it is.

In a category $\mathbf{C}$, a *product of objects* A, B consists of an object P together with two morphisms $\pi_0 : P \to A, \pi_1 : P \to B$. (P, π_0, π_1) is a product of A, B iff for every Z and $\zeta_0 : Z \to A, \zeta_1 : Z \to B$ there is exactly one morphism $(!) : Z \to P$ such that $\zeta_0 = \pi_0 \circ (!)$ and $\zeta_1 = \pi_1 \circ (!)$. Looking at the diagram below we see that the required property of $(!)$, namely that ζ_0, ζ_1 factor through it, is the same as both triangles' commuting:

$$\begin{tikzcd}
& Z \arrow[dl, "\zeta_0"'] \arrow[d, dashed, "(!)"'] \arrow[dr, "\zeta_1"] & \\
A & P \arrow[l, "\pi_0"] \arrow[r, "\pi_1"'] & B
\end{tikzcd}$$

We also simply say that P is the product of A, B if the morphisms π_0, π_1 are clear. When a category $\mathbf{C}$ has a product for every two objects A, B we say that $\mathbf{C}$ *has (binary) products.*

Products, too, are unique up to isomorphism, and we thus write $A \times B$ for 'the' product of

A and B. Consider the diagrams below:

$$\begin{tikzcd}
& P \arrow[dl, "\pi_0"'] \arrow[d, dashed, "(!_0)=\mathrm{Id}_P"] \arrow[dr, "\pi_1"] & \\
A & P \arrow[l, "\pi_0"] \arrow[r, "\pi_1"'] & B
\end{tikzcd}
\qquad
\begin{tikzcd}
& P \arrow[ddl, "\pi_0"'] \arrow[d, dashed, "(!_2)"'] \arrow[ddr, "\pi_1"] & \\
& P' \arrow[dl, "\pi_0'"] \arrow[d, dashed, "(!_1)"'] \arrow[dr, "\pi_1'"] & \\
A & P \arrow[l, "\pi_0"] \arrow[r, "\pi_1"'] & B
\end{tikzcd}$$

$$\begin{tikzcd}
& P' \arrow[dl, "\pi_0'"'] \arrow[d, dashed, "(!_3)=\mathrm{Id}_{P'}"] \arrow[dr, "\pi_1'"] & \\
A & P' \arrow[l, "\pi_0'"] \arrow[r, "\pi_1'"'] & B
\end{tikzcd}
\qquad
\begin{tikzcd}
& P' \arrow[ddl, "\pi_0'"'] \arrow[d, dashed, "(!_1)"'] \arrow[ddr, "\pi_1'"] & \\
& P \arrow[dl, "\pi_0"] \arrow[d, dashed, "(!_2)"'] \arrow[dr, "\pi_1"] & \\
A & P' \arrow[l, "\pi_0'"] \arrow[r, "\pi_1'"'] & B
\end{tikzcd}$$

The left-hand side product diagrams make clear that the identities are the only morphisms to make both triangles in their respective diagrams commute. The right-hand side diagrams show us that the compositions $(!_1) \circ (!_2)$ and $(!_2) \circ (!_1)$ also make those same triangles commute (the outer triangles in their respective diagrams). Therefore these compositions are the identity arrows, showing that P, P' are isomorphic.

Definition 2. *When category* $\mathbf{C}$ *has morphisms* $f : A \to A', g : B \to B'$ *and* $A \overset{\pi_0}{\leftarrow} A \times B \overset{\pi_1}{\to} B$ *and* $A' \overset{\pi_0'}{\leftarrow} A' \times B' \overset{\pi_1'}{\to} B'$ *are products then we define* $(f \times g) : A \times B \to A' \times B'$ *to be the unique morphism that makes the following diagram commute:*

$$\begin{tikzcd}
A \arrow[d, "f"'] & A \times B \arrow[l, "\pi_0"'] \arrow[r, "\pi_1"] \arrow[d, dashed, "(!)=:(f\times g)"] & B \arrow[d, "g"] \\
A' & A' \times B' \arrow[l, "\pi_0'"] \arrow[r, "\pi_1'"'] & B'
\end{tikzcd}$$

Example 3 (Category of sets)**.** *An example of a category is* **Set***, where* $\mathrm{ob}\,\mathbf{Set}$ *is the collection of all sets.* $\mathrm{Hom}(A, B)$ *is the set of all total functions from set* A *to set* B*. The identity morphisms are the identity functions. Total function composition indeed leads to a new total function and is associative.*

Set *has a terminal object, namely any singleton set* $\{*\}$*. Every set* X *has one function to it, namely the function that maps everything to* $*$.

Set *has binary products: the product of* A *and* B *is the cartesian product* $A \times B$ *with projection functions* $\pi_0 : A \times B \to A : (a, b) \mapsto a$ *and* $\pi_1 : A \times B \to B : (a, b) \mapsto b$.

2.2 Cantor's diagonalization argument

In this subsection we establish the concept of a diagonal argument. Our foundations are any reasonably standard, classical (i.e. non-constructive) set theory, e.g. Zermelo-Fraenkel.[1]

[1] The principle of diagonalization arguments themselves is independent of the Axiom of Choice and the Continuum Hypothesis.

The original argument Georg Cantor famously used a diagonal argument to prove that the set ω of natural numbers is "strictly smaller" than the set 2^ω of all subsets of ω (also called the *power set* of ω), i.e. there is no surjective function $\omega \to 2^\omega$.[Geo91] Since 2^ω is equinumerous with the set $\mathbb{R}$ of the real numbers—an argument that we will not expand on here—this allows us to conclude that $\mathbb{R}$ is an uncountably infinite set, i.e. infinite and not in bijective correspondence with ω.

We first provide a formal proof of the statement, inspired by Cantor's diagonal argument.

Theorem 4 (Cantor's theorem for the naturals)**.** *There is no surjective function from ω to 2^ω.*

Proof. Suppose we have a function $i : \omega \to 2^\omega$. We will prove that i is not surjective.

Formally, we construct a $g \in 2^\omega$ such that there is no $n \in \omega$ with $i(n) = g$. To this end, we specify that g contains a given natural number m iff $i(m)$ does *not* contain m. Now there can clearly be no $n \in \omega$ with $i(n) = g$, for that would mean $n \in i(n)$ iff $n \in g$ iff $n \notin i(n)$, which is a contradiction. Therefore, i is not surjective. □

We now rephrase the proof, making the epithet "diagonal" more understandable. We first realize that for any set A, any element X of the power set 2^A can equivalently be seen as a function from A to the set $2 = \{0, 1\}$ of two elements. We let this function map $a \in A$ to 1 iff $a \in X$ and to 0 iff $a \notin X$ and call it the *characteristic function of $X \subset A$*. This means that every element of 2^ω can uniquely be written as a countably infinite string of ones and zeroes, which we visualize as written out horizontally, continuing infinitely to the right. In order to make the diagonalization apparent, we regard $i : \omega \to 2^\omega$ as a countably infinite sequence of such strings, positioned vertically, continuing infinitely downwards. The visualization thus amounts to a quadrant, extending infinitely to the right and bottom. The n-th row of the quadrant represents $i(n)$ and thus features a 1 as the m-th symbol iff $m \in i(n)$ and a 0 iff $m \notin i(n)$. Surjectivity of i is then equivalent to every possible infinite binary string being present as a row in the quadrant.

$$\begin{array}{lllll}
i(0): & i(0)(0), & i(0)(1), & i(0)(2), & \cdots \\
i(1): & i(1)(0), & i(1)(1), & i(1)(2), & \cdots \\
i(2): & i(2)(0), & i(2)(1), & i(2)(2), & \cdots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

Consider again the formal definition of $g \in 2^\omega$ in the proof of Theorem 4. It, too, can be seen as an infinite binary string. In fact, it differs on every position from the string described by the diagonal of the quadrant. Therefore g differs from every row somewhere (namely from the n-th row at its n-th symbol). This means we have constructed a binary string g that is not equal to any row of the quadrant. We conclude that i is not surjective.

Arbitrary domains The argument we just used to prove that 2^ω is strictly greater than ω can be used for any set, not just ω. Cantor claims this in the original paper:

> Dieser Beweis erscheint nicht nur wegen seiner grossen Einfachheit, sondern namentlich auch aus dem Grunde bemerkenswert, weil das darin befolgte Princip sich ohne weiteres auf den allgemeinen Satz ausdehnen lässt, dass die Mächtigkeiten wohldefinirter Mannigfaltigkeiten kein Maximum haben oder, was dasselbe ist, dass jeder gegebenen Mannigfaltigkeit L eine andere M an die Seite gestellt werden kann, welche von stärkerer Mächtigkeit ist als L.[Geo91]

As an example, they repeat the argument for the closed unit interval $I := [0,1]$ instead of ω, i.e. to prove that there is no surjection from I to the set 2^I of all its subsets. We provide a proof for the general case here:

Theorem 5 (Cantor's theorem for arbitrary domains)**.** *Suppose T is a set. There is no surjective function from T to 2^T.*

Proof. Given any function $i : T \to 2^T$, we can define $X := \{t \in T \mid t \notin i(t)\}$ and see that X is not in the range of i. For if some $i(t) = X$, we get that $t \in i(t)$ iff $t \in X$ iff $t \notin i(t)$, which is a contradiction. □

The proof is clearly analogous to that of Theorem 4. Although the quadrant can no longer easily be visualized, the diagonal character of the argument is retained in that X is the complement of the diagonal and thus disagrees with every $i(a)$ on some point, namely on a.

From $T \to 2^T$ to $T \times T \to 2$ Cantor's theorem posits the non-existence of a surjective function from T to 2^T. If we again regard 2^T as the set $T \to 2$ of functions from T to 2, we might ask ourselves how to characterize $T \to (T \to 2)$. In general, for any sets A, B, C we have that $A \to (B \to C)$ is isomorphic to $A \times B \to C$. Given $f : A \to (B \to C)$ we define

$$\begin{aligned} \overline{f} : A \times B &\longrightarrow C \\ (a,b) &\longmapsto \overline{f}((a,b)) = f(a)(b), \end{aligned}$$

and, conversely given $g : A \times B \to C$ we define

$$\begin{aligned} \overline{g} : A &\longrightarrow (B \to C) \\ a &\longmapsto \overline{g}(a) = \text{the function mapping } b \text{ to } g(a,b). \end{aligned}$$

Clearly, by extensionality $\overline{\overline{f}} = f$ because $\overline{\overline{f}}(a)(b) = \overline{f}(a,b) = f(a)(b)$. Similarly, $\overline{\overline{g}} = g$.

We consider to which property of $\overline{i}$ the surjectivity of a function $i : T \to (T \to 2)$ corresponds. It amounts to the property that for all $g : T \to 2$, there is some $t \in T$ such that $\overline{i}(t, -) = g$. We refer to this property as *representability*:

Definition 6. *Assume functions $f : A \times B \to C$ and $g : B \to C$. We say that g is* represented *by $a \in A$ in f iff $g = f(a, -)$. We say that g is* representable in *f iff there exists some $a \in A$ that represents g in f.*

Cantor's theorem can thus be rephrased:

Theorem 7 (Cantor's theorem with representable functions)**.** *Suppose T is a set. For every function $f : T \times T \to 2$, there is some $g : T \to 2$ that is not representable in f.*

Fixpoints & arbitrary codomains We consider again the definitions of X and i in the proof of Theorem 5. If we choose to identify X and every $i(t)$ with their characteristic functions from T to $2 = \{0, 1\}$, we see that $X(t) = 1 - i(t)(t) = 1 - \overline{i}(t,t)$ for every $t \in T$. Then X can be seen as the composition of three functions: $X = \alpha \circ \overline{i} \circ \Delta$ where $\Delta : T \to T \times T : \Delta(t) := (t,t)$ is the diagonal function and $\alpha : 2 \to 2 : \alpha(x) := 1 - x$ switches its inputs around. Cantor's theorem thus says that for every $f : T \times T \to 2$, we can define a $g : T \to 2$ as in the following commutative diagram in **Set**, such that g is not representable in f:

$$\begin{tikzcd}
T \times T \arrow[r, "f"] & 2 \arrow[d, "\alpha"] \\
T \arrow[u, "\Delta"] \arrow[r, "g"'] & 2
\end{tikzcd}$$

Looking at the proof of Theorem 5, we see that the argument still goes through if we substitute an arbitrary set Y for 2, as long as α has no fixpoints (i.e. there is no $y \in Y$ such that $\alpha(y) = y$):

$$\begin{tikzcd}
T \times T \arrow[r, "f"] & Y \arrow[d, "\alpha"] \\
T \arrow[u, "\Delta"] \arrow[r, "g"'] & Y
\end{tikzcd}$$

This is a further generalization than is done in the original Cantor paper.[Geo91] We therefore call it the *generalized Cantor theorem* or *GCT*.

Theorem 8 (Cantor's theorem for arbitrary sets)**.** *Suppose T, Y are sets. Suppose $\alpha : Y \to Y$ is a function without fixpoints. Then for every function $f : T \times T \to Y$, there is some $g : T \to Y$ that is not representable in f. Specifically, $g := \alpha \circ f \circ \Delta$ is not representable in f.*

Proof. Let T, Y, α, f be as in the theorem. Define $g := \alpha \circ f \circ \Delta$. Now g cannot be representable in f. For suppose that g is represented in f by some $t \in T$. Then $g(t) = f(t, t)$, but by definition of g we also have $g(t) = \alpha(f(t, t))$. Since α is assumed to have no fixpoints, this is a contradiction. □

2.3 Cantor generalized to categories

In Section 2.2 we have worked, mostly implicitly, in the category **Set** of sets and total functions. It exhibits many properties that not every category has. In this subsection we present Yanofsky's reformulation of Cantor's diagonal argument in the language of categories.

Our latest version of Cantor's theorem so far, Theorem 8, relies on several set-theoretic notions. First, there is the cartesian product and the diagonal function $\Delta : T \to T \times T$. Second, the function α is required to be free of fixpoints. Third, g is said not to be representable in f. We now seek purely categorical rephrasings of these notions that generalize their originals in **Set**. We follow [Yan22], who based themself on [Law69].

Cartesian product and diagonal function As long as the relevant category has binary products, the role of the diagonal function $\Delta : T \to T \times T$ can be taken by a morphism $\Delta : T \to T \times T$.

Fixpoints A function $\alpha : Y \to Y$ has a fixpoint iff there is some $y \in Y$ such that $\alpha(y) = y$, by definition. We realize that Y is in bijective correspondence with the set $1 \to Y$ of functions from any singleton set $1 := \{*\}$ to Y: every $y \in Y$ is associated with the function $\overline{y}$ that maps $*$ to y and vice versa. As long as this bijection holds, we choose to identify every element $y \in Y$ with its morphism $\overline{y} : 1 \to Y$. We say that a category where this correspondence is bijective for all objects (which necessarily are sets) *has elements*. We deliberately keep the definition vague and will readdress the issue in Section 4. Clearly, **Set** has elements. We can therefore define fixpoints categorically as follows:

Definition 9. *Let* **C** *be a category with elements and with a terminal object. A morphism* $A \xrightarrow{a} A'$ *is said to have a fixpoint iff there is some* $1 \xrightarrow{p} A$ *such that* $a \circ p = p$.

Representability To reformulate Definition 6 in category-theoretic language we must generalize the notion $f(a,-) = g$ for $f : A \times B \to C$, $g : B \to C$, $a \in A$. We see that the function $f(a,-) = (b \mapsto f(a,b)) : B \to C$ is a composition $f \circ (a,-)$ of $f : A \times B \to C$ with the function $(a,-) = (b \mapsto (a,b)) : B \to A \times B$. If we assume the ambient category has elements, binary products and a terminal object we can reformulate $(a,-)$ categorically as $(\overline{a} \times \mathrm{Id}_B) \circ \imath$, where $\imath : B \to 1 \times B$ is the unique arrow $(!_0)$ from this diagram:

$$\begin{tikzcd}
 & 1 \times B \arrow[ddl, "\pi_0=(!_{1\times B})"'] \arrow[ddr, "\pi_1"] \arrow[d, dashed, "(!_1)=:\pi_1"] & \\
 & B \arrow[dl, "(!_B)"'] \arrow[dr, "\mathrm{Id}_B"] \arrow[d, dashed, "(!_0)=:\imath"'] & \\
1 & 1 \times B \arrow[l, "\pi_0=(!_{1\times B})"] \arrow[r, "\pi_1"'] & B
\end{tikzcd}$$

It is not difficult to show that $\imath$ is an isomorphism (with π_1 as the opposite morphism). The generalization of $f(a,-)$ then becomes $f \circ (\overline{a} \times \mathrm{Id}_B) \circ \imath$.

Definition 10. *Let* **C** *be a category with elements, binary products and a terminal object. Let* $A \times B \xrightarrow{f} C$ *and* $B \xrightarrow{g} C$ *be morphisms. Then* g is represented in f by $p : 1 \to A$ *iff* $f \circ (p \times id_B) \circ \imath = g$, *where* $\imath : B \to 1 \times B$ *is the obvious isomorphism. We say that* g is representable in f *iff* g *is represented in* f *by some* $p : 1 \to A$.

We have now generalized all the necessary concepts to conclude Yanofsky's categorical version of the generalized Cantor theorem:[Yan22]

Theorem 11 (Yanofsky's GCT)**.** *Let* $\mathcal{C}$ *be a category with elements, binary products and a terminal object. Let* T, Y *be objects in* $\mathcal{C}$. *Let* $f : T \times T \to Y$ *and* $\alpha : Y \to Y$ *be morphisms in* $\mathcal{C}$ *and let* α *be without fixpoints. Then there exists a morphism* $g : T \to Y$ *that is not representable in* f. *Specifically,* $g := \alpha \circ f \circ \Delta$ *is not representable in* f.

Proof. Let $\mathcal{C}, T, Y, f, \alpha$ be as in the theorem. Define $g := \alpha \circ f \circ \Delta$. Suppose that g is represented by $1 \xrightarrow{t} T$ in f, i.e. $\alpha \circ f \circ \Delta = f \circ (t \times \mathrm{Id}_B) \circ \imath$. Suppose for the sake of a contradiction that g is representable in f by some $1 \xrightarrow{t} T$. Then of course $\alpha \circ f \circ \Delta \circ t = f \circ (t \times \mathrm{Id}_B) \circ \imath \circ t$. We claim that $(t \times \mathrm{Id}_B) \circ \imath \circ t = \Delta \circ t$, which will imply $\alpha \circ f \circ \Delta \circ t = f \circ (t \times \mathrm{Id}_B) \circ \imath \circ t = f \circ \Delta \circ t$. This means by definition that $f \circ \Delta \circ t$ is a fixpoint of α.

We now prove the claim $(t \times \mathrm{Id}_B) \circ \imath \circ t = \Delta \circ t$. We consider the product $T \xleftarrow{\pi_0} T \times T \xrightarrow{\pi_1} T$ and show that $T \xleftarrow{t} 1$ and $1 \xrightarrow{t} T$ factor through both $(t \times \mathrm{Id}_B) \circ \imath \circ t : 1 \to T \times T$ and $\Delta \circ t : 1 \to T \times T$. By definition of product there can only be one such morphism and thus they are equal.

On the one hand we see from chasing this diagram

$$\begin{tikzcd}
& 1 \arrow[d, "t"] & \\
& T \arrow[dl, "(!_T)"'] \arrow[d, "\imath"'] \arrow[dr, "\mathrm{Id}_T"] & \\
1 \arrow[d, "t"'] & 1 \times T \arrow[l, "\pi_0=(!_1)"] \arrow[r, "\pi_1"'] \arrow[d, "t\times \mathrm{Id}_T"'] & T \arrow[d, "\mathrm{Id}_T"] \\
T & T \times T \arrow[l, "\pi_0"'] \arrow[r, "\pi_1"] & T
\end{tikzcd}$$

that $\pi_0 \circ (t \times \mathrm{Id}_B) \circ \imath \circ t = t \circ \pi_0 \circ \imath \circ t = t \circ (!_T) \circ t = t$ and $\pi_1 \circ (t \times \mathrm{Id}_T) \circ \imath t = \mathrm{Id}_T \circ \pi_1 \circ \imath \circ t = \mathrm{Id}_T \circ \mathrm{Id}_T \circ t = t$, where in both cases the three equalities can be found by definition of $(t \times \mathrm{Id}_B)$, $\imath$ and terminal object, respectively.

On the other hand we see from this diagram

$$\begin{tikzcd}
& 1 \arrow[d, "t"] & \\
& T \arrow[dl, "\mathrm{Id}_T"'] \arrow[d, "\Delta"'] \arrow[dr, "\mathrm{Id}_T"] & \\
T & T \times T \arrow[l, "\pi_0"] \arrow[r, "\pi_1"'] & T
\end{tikzcd}$$

that $\pi_0 \circ \Delta \circ t = t$ and $\pi_1 \circ \Delta \circ t$. Since t and t factor through $(t \times \mathrm{Id}_T) \circ \imath \circ t$ as well as $\Delta \circ t$, we conclude that these latter two morphisms must be equal by definition of product. ☐

By contraposition we also obtain the following:

Corollary 12. *Let $\mathcal{C}$ be a category with elements, binary products and a terminal object. Let T, Y be objects in $\mathcal{C}$. Let $f : T \times T \to Y$ and $\alpha : Y \to Y$ be morphisms in $\mathcal{C}$. Suppose $g := \alpha \circ f \circ \Delta$ is represented in f by $1 \xrightarrow{t} T$. Then α has a fixpoint, namely $f \circ \Delta \circ t$.*

We will be referring to this corollary very often and will sometimes simply identify it with Yanofsky's GCT.

3 The Ackermann function is not primitive recursive

Now that we have established the categorical version of Yanofsky's GCT it is time to apply it. We will attempt to use it to prove mathematical results that are usually proved using arguments that seem diagonal.

3.1 Prerequisites

Our first goal is to work towards a suitable definition of *computable functions*, for which we follow Cutland.[Cut80] We first put forward an abstract, mathematical model of a computer called an *unlimited register machine* (URM). The URM model will delineate a class of functions which we will call *URM-computable.*

After this, we will introduce a seemingly completely different notion, that of *μ-recursive* functions. It turns out that these two and many other definitions result in the same class of functions, which we will therefore simply call *computable functions.*

Unlimited register machine The unlimited register machine is postulated to have an infinite number of *registers*, exhaustively numbered $R_1, R_2, R_3, \ldots$, all of which hold a natural number (including zero) at any given time. An infinite sequence $r_1, r_2, r_3, \ldots$ of natural numbers thus represents a possible combined state of the registers. We say that the URM is *in configuration* $r_1, r_2, \ldots$ when R_1 holds value r_1 and R_2 holds r_2, etc.

Further, there is a fixed set $\mathscr{I}$ of URM *instructions* (to be defined in a moment). A finite list of instructions $I_1, I_2, \ldots, I_k$ is called a *program*. We refer to the set of all programs as $\mathscr{P}$. In order for the URM to perform a computation, we must supply it with an initial configuration $r_1, r_2, \ldots$ and a program $I_1, \ldots, I_k$. For brevity, we refer to a configuration $r_1, \ldots, r_l, 0, 0, 0, \ldots$ as $r_1, \ldots, r_l$. The URM executes the instructions one by one, starting with I_1. Every instruction can prompt the URM to change the contents of some registers and tells it which instruction to execute next.

Below is the exhaustive list of all possible instructions. We refer to the configurations before and after execution of the relevant instruction as $r_1, r_2, \ldots$ and $r'_1, r'_2, \ldots$, respectively.

Zero For every register R_n, there is an instruction $Z(n)$. It sets the value of R_n to zero. Formally, $r'_n = 0$ and $r'_k = r_k$ for all $k \neq n$. Execution continues with the next instruction.

Successor For every register R_n, there is an instruction $S(n)$. It increases the value of R_n by one. Formally, $r'_n = r_n + 1$ and $r'_k = r_k$ for all $k \neq n$. Execution continues with the next instruction.

Transfer For every two registers R_m, R_n, there is an instruction $T(m, n)$. It overwrites R_n with the value of R_m. Formally, $r'_n = r_m$ and $r'_k = r_k$ for all $k \neq n$. Execution continues with the next instruction.

Jump For every two registers R_m, R_n and every natural number i, there is an instruction $J(m, n, i)$. It does not alter the registers. If $r_m \neq r_n$, execution continues with the next instruction. If $r_m = r_n$, execution continues with instruction number i.

We have now introduced all the machinery of the URM model. It can happen that the URM, when executing program $P = I_1, \ldots, I_k$ on initial configuration $\mathbf{a} = a_1, \ldots, a_l$, tries to execute a non-existent instruction, i.e. one with a higher number than k. This can happen because of a jump or if I_k is executed and it does not evoke a jump. When this happens, the URM's execution *terminates* or *halts*, we say that $P(\mathbf{a})$ *converges* and we write $P(\mathbf{a}) \downarrow$.

So far, we have referred to an execution by the URM as a computation. What, then, does the URM compute? We choose to regard the contents of R_1 after execution of program P on initial configuration $\mathbf{a}$ as the result of the computation $P(\mathbf{a})$. If we refer to contents of R_1 after termination of $P(\mathbf{a})$ as b, we say that $P(\mathbf{a})$ *converges to* b and write $P(\mathbf{a}) \downarrow b$.

It can also happen, of course, that the execution of $P(\mathbf{a})$ does not terminate. In such cases, there is no such thing as the contents of R_1 at 'after execution'. We then say that $P(\mathbf{a})$ *diverges* and write $P(\mathbf{a}) \uparrow$.

Take, for example, the program $P = J(0, 1, 1)$. It does terminate for input vectors of two different numbers; e.g. $P(5, 6) \downarrow 5$ because the jump is not executed. For equal numbers, however, the execution does not converge since the first and only instruction causes the URM to jump back to the first instruction and repeat it infinitely; e.g. $P(5, 5) \uparrow$.

URM computation allows us to define a notion of computability for functions on the natural numbers:

Definition 13. *Suppose $m \geq 1$ and $f : \mathbb{N}^m \rightharpoonup \mathbb{N}$ is an m-ary partial function on the natural numbers. We say that f is* URM-computable *iff there exists a program P such that: for all $\mathbf{x} \in \mathbb{N}^m$ in the domain of f, $P(\mathbf{x}) \downarrow f(\mathbf{x})$, and for all $\mathbf{x} \in \mathbb{N}^m$ not in the domain of f, $P(\mathbf{x}) \uparrow$.*

We let $\mathscr{C}^m$ denote the set of all m-ary URM-computable functions and let $\mathscr{C}$ denote the set of all URM-computable functions of any arity.

μ-recursive functions We now take a very different approach to the definition of computability of m-ary functions on the naturals. The idea is to assume that a few very basic functions are computable and also that three specific operations on functions preserve computability. In fact, we define the class of *μ-recursive* functions inductively in this fashion:[2]

Zero function We define a unary function $Z : \mathbb{N} \to \mathbb{N}$ that maps every input to 0.

Successor function We define a unary function $S : \mathbb{N} \to \mathbb{N}$ that maps x to $x + 1$.

Projection functions For every arity m and every natural number $i \leq m$, we define an m-ary projection function $U_i^m : \mathbb{N}^m \to \mathbb{N}$ that maps $(x_1, \dots, x_m)$ to x_m.

Substitution When $f : \mathbb{N}^m \rightharpoonup \mathbb{N}$ is an m-ary partial function and $g_1, \dots, g_m : \mathbb{N}^l \rightharpoonup \mathbb{N}$ are m partial functions of arity l, let $f \circ (g_1, \dots, g_m) : \mathbb{N}^l \rightharpoonup \mathbb{N}$ be an l-ary partial function that maps $\mathbf{x} = (x_1, \dots, x_l)$ to $f(g_1(\mathbf{x}), \dots, g_m(\mathbf{x}))$. (It is undefined whenever any subexpression as presented here is undefined). Note that when $m = 1$, substitution as defined here coincides with the standard function composition.

Recursion When $f : \mathbb{N}^m \rightharpoonup \mathbb{N}$ is an m-ary partial function and $g : \mathbb{N}^{m+2} \rightharpoonup \mathbb{N}$ is an $m + 2$-ary partial function, let $R_{f,g} : \mathbb{N}^{m+1} \rightharpoonup \mathbb{N}$ be an $m + 1$-ary partial function that maps $(\mathbf{x}, 0)$ to $f(\mathbf{x})$ and $(\mathbf{x}, y + 1)$ to $g(\mathbf{x}, y, R_{f,g}(\mathbf{x}, y))$. (It is undefined whenever any subexpression as presented here is undefined). This construction appears to be self-referential. However, we note that the use of $R_{f,g}$ in order to define $R_{f,g}$ is well-founded, since it is only ever used for strictly lower values of y. This construction provides us with a formal version of the computer programming notion of well-founded, non-mutual recurson.

Bounded minimization When $f : \mathbb{N}^{m+1} \rightharpoonup \mathbb{N}$ is an $m + 1$-ary partial function, let $\mu_f^<$ be a function that maps $(y, x_1, \dots, x_m)$ to the smallest z that is less than y such that $f(z, \mathbf{x}) = 0$ and $f(z', \mathbf{x})$ is defined for all $z' < z$, if such a z exists. $\mu_f^<(y, \mathbf{x})$ is undefined if the sequence $f(0, \mathbf{x}), f(1, \mathbf{x}), \dots, f(y - 1, \mathbf{x})$ features an undefined element without reaching 0 before. If the sequence is everywhere defined but never reaches 0, then $\mu_f^<(y, \mathbf{x})$ is defined to be y. $\mu_f^<(y, \mathbf{x})$ is of course also undefined if any of $y, x_1, \dots, x_m$ is undefined.

Note that bounded minimization strongly resembles the `for` loop from iterative programming languages. We can think of a variable z' iterating from 0 to a predetermined value y. At every iteration, $f(z', \mathbf{x})$ is calculated. If it is 0, the loop is exited and we report z'. If it is not 0, we increase z' and continue with the next iteration of the loop. If the calculation of $f(z, \mathbf{x})$ does not finish (i.e. it is undefined), the loop execution also never finishes (i.e. its value is undefined). If the loop finishes through exhaustion of the iterations, we report y to indicate this.

Unbounded minimization When $f : \mathbb{N}^{m+1} \rightharpoonup \mathbb{N}$ is an $m + 1$-ary partial function, let μ_f be a function that maps $\mathbf{x}$ to the smallest z such that $f(z, \mathbf{x}) = 0$ and $f(z', \mathbf{x})$ is defined for all $z' < z$, if such a z exists. $\mu_f(\mathbf{x})$ is undefined if the sequence $f(0, \mathbf{x}), f(1, \mathbf{x}), f(2, \mathbf{x}), \dots$ features an undefined element before it reaches 0 or the sequence never reaches 0. $\mu_f^<(y, \mathbf{x})$ is of course also undefined if any of $x_1, \dots, x_m$ is undefined.

[2]Note that what we call μ-recursive is called *partial recursive* by Cutland.[Cut80] Both refer to the functions in Cutland's $\mathscr{R}$.

Note that unbounded minimization strongly resembles the `while` loop from iterative programming languages. We can think of a variable z' iterating up from 0 indefinitely until the loop is broken. The procedure of calculation is similar to the `for` loop of unbounded minimization. The only difference is that the loop never finishes through exhaustion. In case the loop continues forever, its value is undefined.

We have now introduced all the relevant concepts for a second definition of computability:

Definition 14. *We define the set $\mathscr{PR}$ of* primitive recursive functions *to be the smallest set of partial functions including the zero function Z, the successor function S, all projection functions U_i^m and closed under substitution $f \circ \mathbf{g}$, recursion $R_{f,g}$ and bounded minimization $\mu_f^<$.*

We define the set $\mathscr{R}$ of partial recursive*,* μ-recursive*,* general recursive *or simply* recursive *functions to be the smallest set of partial functions including the zero function Z, the successor function S, all projection functions U_i^m and closed under substitution $f \circ \mathbf{g}$, recursion $R_{f,g}$ and unbounded minimization μ_f.*

Most of the time we use the term "function" to refer to a function with any power of $\mathbb{N}$ as the domain and $\mathbb{N}$ as the codomain. However, sometimes the codomain will also be a higher power of $\mathbb{N}$. Any function from $\mathbb{N}^m$ to $\mathbb{N}^n$ can be thought of as an n-tuple of functions from $\mathbb{N}^m$ to $\mathbb{N}$: for each $f : \mathbb{N}^m \to \mathbb{N}^n$ there are $f_1, \ldots, f_n : \mathbb{N}^m \to \mathbb{N}$ such that $f(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))$. The definitions of primitive recursive function and partial recursive function are then easily extended: $f : \mathbb{N}^m \to \mathbb{N}^n$ is primitive recursive (resp. partial recursive) iff all functions in the n-tuple are.

Any predicate on a power $\mathbb{N}^m$ of the natural numbers is said to be *decidable* if its characteristic function is computable, where the characteristic function of a predicate P maps all $\mathbf{m} \in \mathbb{N}^m$ that satisfy P to 1 and all $\mathbf{m} \in \mathbb{N}^m$ that do not satisfy P to 0.

Note that primitive recursive functions are always total, since the basic functions are total and its three constructions $\circ$, R and $\mu^<$ preserve totality.[Cut80] Many ordinary arithmetic operations and convenient constructions can be shown to be primitive recursive, e.g. addition $(m, n) \mapsto m + n$, multiplication and exponentiation. and function definition by parts where the case distinction is based on simple criteria like $m < n$.[Men10]

The relationship between the computability notions We now examine the relationship between the URM-computable functions $\mathscr{C}$, the primitive recursive functions $\mathscr{PR}$ and the general recursive functions $\mathscr{R}$.

To begin, we can convince ourselves that unbounded minimization is at least as strong as bounded minimization. Every bounded minimization

$$\mu_f^<(y, \mathbf{x}), \quad \text{where } f : \mathbb{N}^m \rightharpoonup \mathbb{N}$$

can be simulated using unbounded minimization:[3]

$$\mu_{f'}(y, \mathbf{x}), \quad \text{where } f' : \mathbb{N}^{m+1} \rightharpoonup \mathbb{N} : (z, \mathbf{x}) \mapsto \begin{cases} 0 & z \geq y \\ f(z, \mathbf{x}) & z < y \end{cases}.$$

Therefore, $\mathscr{R}$ is also closed under bounded minimization and with that under all the things on which $\mathscr{PR}$ is defined inductively. We can conclude that all primitive recursive functions are general recursive: $\mathscr{PR} \subseteq \mathscr{R}$.

We naturally ask ourselves if, conversely, $\mathscr{R} \subseteq \mathscr{PR}$. Much of the rest of Section 3 is dedicated to a proof that this is not the case, i.e. there exists a function (e.g. the Ackermann function)

[3]Such a case distinction can be made in general recursive functions using recursion. (See Cutland [Cut80].)

which is general recursive but not primitive recursive. Unbounded minimization thus truly adds computational power to the model.

Perhaps more surprisingly, a meaningful comparison can be made between $\mathscr{C}$ and $\mathscr{R}$. They are equal! Cutland shows how these two very different characterizations of computable functions end up defining the same functions.[Cut80] In short, they prove that $\mathscr{R} \subseteq \mathscr{C}$ inductively, i.e. by showing that $\mathscr{C}$ contains all the basic functions of $\mathscr{R}$ and is closed under the same operations as $\mathscr{R}$. They do this explicitly, by constructing URM-programs for Z, S, U_i^m and for $f \circ (g_1, \ldots, g_m)$, $R_{f,g}$ and μ_f given programs for $f, g, g_1, \ldots, g_m$.

They prove that $\mathscr{C} \subseteq \mathscr{R}$ by simulating the execution of a program P of any URM-computable $f \in \mathscr{C}$ with recursive functions. They define helper functions j and c. Loosely, $j(t, \mathbf{x})$ equals the number of the instruction to be executed after t steps of computing $P(\mathbf{x})$, or 0 if the machine has already halted by then. $c(t, \mathbf{x})$ equals the contents of R_1 after t steps of computing $P(\mathbf{x})$. We then see that $f(\mathbf{x}) \simeq c(\mu_j(\mathbf{x}), \mathbf{x})$. The crux of the proof is that they show j and c to be general recursive.

It turns out that many more notions of computability are equivalent. In 1936, Alan Turing proposed the 'automatic machine' (now called 'Turing machine').[Tur37] Just like for the unlimited register machine, a convention is established to determine the function which is being computed by a Turing machine. Around the same time, Alonzo Church proposed the 'lambda calculus' formalism,[Chu36] which now serves as the theoretical framework for functional programming languages like ML and Haskell. Again, a convention is established to interpret mathematical functions in the formalism and vice versa.

It has been proven, although we do not elaborate on it here, that for any function, computability by Turing machines, computability in the lambda calculus, computability by the URM and general recursivity are all equivalent. It therefore makes sense to speak only of *computability.* There is an informal statement in computability theory, the *Church-Turing thesis*, saying that every function that can intuitively be said to be calculated systematically, is in fact computable in the above sense. In other words, all sufficiently strong models of computation are equivalent.

3.2 There exist recursive, non-primitive recursive functions

We now set out to prove that $\mathscr{R} \nsubseteq \mathscr{PR}$, i.e. there exists a recursive function that is not primitive recursive. The general strategy for this will be to construct an instance of Yanofsky's GCT where we choose f such that representability in f is equivalent to primitive recursivity. We will make sure that α has no fixpoints and conclude that there is a non-representable and thus non-primitive recursive function. We must make sure that this non-representable morphism is recursive, i.e. computable.

Let **TComp** be the category of the natural numbers and its finite products ($\mathbb{N}^0$, $\mathbb{N}^1$, $\mathbb{N}^2$ etc.), where the morphisms are the *total* computable functions. We say that $\mathbb{N}^0$ is a singleton $\{*\}$ and expand the definition such that all total functions from or to $\mathbb{N}^0$ are computable. Totality is important for the GCT theorem to go through. By the Church-Turing thesis, the precise formalism for computability is not important, but we may say that a morphism $\mathbb{N}^m \xrightarrow{f} \mathbb{N}^n$ is a function computed by an algorithm with m inputs and n outputs (or an n-tuple of functions from $\mathbb{N}^m$ to $\mathbb{N}$). The role of identity morphisms is fulfilled by the identity functions $\mathbb{N}^m \xrightarrow{\mathrm{Id}} \mathbb{N}^m$. Since function composition is associative and preserves computability and totality it is thus clear that **TComp** is indeed a category.

TComp has a terminal object, namely $\mathbb{N}^0 = \{*\}$: from every object $\mathbb{N}^m$ there goes exactly one morphism to $\mathbb{N}^0$, namely the function that maps everything to $*$. It is clearly computable. Furthermore, there is a clear bijection between $\mathbb{N}^m$ and $\mathrm{Hom}_{\mathbf{TComp}}(\mathbb{N}^0, \mathbb{N}^m)$: $\mathbf{n} = (n_1, \ldots, n_m)$ corresponds to the function $\overline{\mathbf{n}}$ which maps $*$ to $\mathbf{n}$. Therefore **TComp** has elements.

We also clearly see that **TComp** has products. The product of $\mathbb{N}^N$ and $\mathbb{N}^M$ is simply $\mathbb{N}^{N+M}$ with $\pi_0(n_1, \dots, n_N, n_{N+1}, \dots, n_{N+M}) := (n_1, \dots, n_N)$ and $\pi_1(n_1, \dots, n_N, n_{N+1}, \dots, n_{N+M}) := (n_{N+1}, \dots, n_{N+M})$. We do not prove this in more detail.

We are looking for a total computable function $f : \mathbb{N}^2 \to \mathbb{N}$ such that primitive recursivity of a total unary function $g : \mathbb{N} \to \mathbb{N}$ is equivalent to there being some m such that $g = f(m, -)$. We call such an f *primitive universal*:[4]

Definition 15. *A total computable function $f : \mathbb{N}^2 \to \mathbb{N}$ is* primitive universal *iff for each total unary function $g : \mathbb{N} \to \mathbb{N}$, g is primitive recursive iff there is an m such that $g = f(m, -)$.*

Proposition 16. *There exists a primitive universal function f such that for each primitive recursive $g : \mathbb{N} \to \mathbb{N}$ there are infinitely many m such that $g = f(m, -)$.*

Proof sketch. We choose some finite alphabet Σ that can describe the primitive recursive functions and let Σ^* denote the set of finite strings over Σ. Σ could, for example, contain `Z`, `S`, `U`, $\circ$, `R`, `mu<`, `0`, `1`, …, `9` and some punctuation for function application and subscripts, e.g. `mu<_Z(0, 0)` for $\mu_Z^{\lessgtr}(0, 0)$. Equivalently, it could be the alphabet of some `for` loop language. We choose some bijective encoding of Σ^* into $\mathbb{N}$. (See [Cut80] for an idea on what such an encoding could look like; it is a recurring concept.) We may speak of a function or program working with a *string* rather than its encoding.

Though there are infinitely many valid programs, it might not be that every string in Σ^* is a valid program. (For example, in the language of primitive recursive functions, `mu<_Z(0, 0)` is a valid program and `Z0(UR9` is not.) The language must be chosen such that it is decidable whether (the encoding of) a given string is (the encoding of) a valid program. Σ^* is countably infinite so its strict subset V of valid programs is also countably infinite.

We now describe a program P_f that computes a satisfactory f. When given (m, n), P_f starts enumerating all strings in Σ^*. Whenever it encounters a valid program string it increases a counter that started at 0. As soon as the counter reaches m, P_f stops enumerating and considers the program string at hand. P_f then simulates an application of the program to n and outputs the result. This can indeed be done systematically; such simulation is essentially what any programming language interpreter does. (This is perhaps easiest to see when we choose the `for` loop language for program representation.)

Fix an arbitrary primitive recursive, unary function $g : \mathbb{N} \to \mathbb{N}$. By definition, g is described by some program string. Let this string be the m-th valid program string as iterated over by P_f. Then, by construction of P_f, for each n we have $g(n) = f(m, n)$, amounting to $g = f(m, -)$.

Conversely, fix an arbitrary total unary function $g : \mathbb{N} \to \mathbb{N}$ and m such that $g = f(m, -)$. By construction, g is computed by the m-th valid program as iterated over by P_f. Every program in the language computes a primitive recursive function and therefore g is primitive recursive.

Suppose that a primitive recursive function g is computed by program P in the language. Then we can trivially modify P into a new program string. In the language of primitive recursive functions we can turn P into $U_1^1 \circ P$. In a for loop language we might add a useless statement, e.g. `x := x`. The new program evidently computes the same function g. We can repeat this arbitrarily often and thus there are infinitely many m such that $g = f(m, -)$. $\square$

We now have everything in place to apply Cantor's theorem:

Proposition 17. *There exists a recursive function on $\mathbb{N}$ which is not primitive recursive.*

[4]Not to be confused with the existing notion of universal function, which is an analogon for recursive functions. We will touch on this subject in Section 4.

Proof. We apply Yanofsky's GCT (Theorem 11) in **TComp**, which has elements, binary products and a terminal object. We supply a primitive universal function $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and any total unary function α without fixpoints, e.g. $\alpha(n) := n + 1$. The theorem tells us that $g := \alpha \circ f \circ \Delta$ is not representable in f. Nevertheless g is a morphism in **TComp** and thus a total computable function. Clearly, α and the diagonal function Δ and thus also g are computable. Since α has no fixpoints Cantor's theorem tells us that g is not representable in f. Then g is not primitive recursive, by definition of f. $\square$

3.3 Ackermann is not primitive recursive

In Section 3.2, we asked ourselves whether all recursive functions are primitive recursive. We got a negative answer from Yanofsky's GCT. However, the counterexample g it provided was rather abstract: $g(n)$ is one more than the n-th primitive recursive function (according to some enumeration of them) applied to n. In the early years of computability theory, a very concrete function emerged that was proven to be a counterexample as well: the *Ackermann function.*[5]

For the rest of Section 3, we investigate the freedom that Yanofsky's GCT gives us. We want to know if it can produce precisely the Ackermann function as non-primitive recursive counterexample.

Definition 18 (Robinson's Ackermann function)**.** *We define the following binary recursive function:*

$$\begin{array}{rccccl} \psi : & & \mathbb{N} \times \mathbb{N} & & & \to \mathbb{N} \\ & (& 0 & , & n &) \mapsto n+1 \\ & (& m+1 & , & 0 &) \mapsto \psi(m,1) \\ & (& m+1 & , & n+1 &) \mapsto \psi(m, \psi(m+1, n)) \end{array}$$

ψ is clearly well-defined: it only makes use of values of ψ for inputs strictly lower in a lexicographical ordering. ψ is also recursive: it is not difficult to imagine a computer program using `while` *loops that executes ψ. (See Section A for a full program.)*

For convenience, we also define the following unary recursive functions:

$$\begin{aligned} A : \mathbb{N} &\to \mathbb{N} \\ n &\mapsto \psi(n,n) \end{aligned}$$

$$\begin{aligned} A' : \mathbb{N} &\to \mathbb{N} \\ n &\mapsto \psi(n,n) + 1 \end{aligned}$$

The statement that we wish to prove is the following:

Theorem 19 (The Ackermann function is not primitive recursive)**.** *ψ is not primitive recursive.*

Proofs of Theorem 19 often make use of the helper functions A and/or A'. We see that primitive recursivity of ψ would imply primitive recursivity of A, e.g., formally, $A := \psi \circ (U_1^1, U_1^1)$. Primitive recursivity of A and A' are clearly equivalent. By contraposition, we see that non-primitive recursivity of A (or A') is sufficient for non-primitive recursivity of ψ.

A common proof strategy for Theorem 19 is to make use of ψ's strict monotonicity in both arguments: if $m < m'$ and $n < n'$ then $\psi(m,n) < \psi(m',n)$ and $\psi(m,n) < \psi(m,n')$.[Men10, exercise 5.40(j)] That is then used to prove the following lemma:

Lemma 20 (Majorization lemma)**.** *For every primitive recursive function $f : \mathbb{N}^l \to \mathbb{N}$, there is some m such that for all inputs $(x_1, \ldots, x_l)$ to f, we have $f(x_1, \ldots, x_l) < \psi(m, \max(x_1, \ldots, x_l))$.*

[5] It is ordinarily called the *Ackermann function* after Wilhelm Ackermann.[Ack28] We follow this convention, although we use a more streamlined version better attributed to Raphael Robinson or Rózsa Péter.[Rob48; Pét35]

Proof sketch. By induction on the definition of primitive recursive functions, i.e. we prove the property holds when f is Z, S, U_l^i and that it is closed under substitution, recursion and bounded minimization. □

From this, Theorem 19 follows as a corollary by contradiction:

Corollary 21 (Theorem 19 from Majorization lemma)**.** *ψ is not primitive recursive.*

Proof. Suppose that ψ is primitive recursive. Then A' is primitive recursive. By the majorization lemma we have some m such that for all n, $A'(n) = \psi(n,n) + 1 < \psi(m, \max(n,n)) = \psi(m,n)$. But this leads to the contradiction that $\psi(m,m)+1 < \psi(m,m)$. Therefore, ψ cannot be primitive recursive. □

We see that this common proof strategy makes no direct use of a diagonal argument. However, Yanofsky has suggested that "Ackermann's function "diagonalizes-out" of primitive recursive functions."[Yan03] We seek to make this precise.

Ackermann is not primitive recursive, a diagonal attempt We would like to know if Yanofsky's GCT allows us to conclude directly that A (and thereby ψ) is not primitive recursive. We choose again a primitive universal f such that primitive recursivity is equivalent to representability in f. We will be working in the category **Set** of sets, where the morphisms are functions between the sets. Recall that Yanofsky's GCT allows us to conclude that $g := \alpha \circ f \circ \Delta$ is not representable in a given f as long as α has no fixpoints. We therefore must examine whether g can be made to be exactly A by providing a suitable α.

Note that $g = A$ would imply the following:

$$\text{for all } m, \quad \alpha(f(m,m)) = g(m) = A(m) = \psi(m,m). \tag{1}$$

We can already see, though, that such an α might be overconstrained. It must map every $f(m,m)$ to $\psi(m,m)$. But in doing so it does not—so to speak—"know" which primitive recursive program P_m^p has been run on which input m to obtain α's input $f(m,m)$. There might very well be some $m' \neq m$ which happens to yield the same value: $f(m',m') = f(m,m)$. Then α must of course map both these equal values to the same value and hence $f(m',m') = \psi(m,m)$. The constraint on α, however, still requires $\alpha(f(m',m')) = \psi(m',m')$. We conclude that whenever $f(m,m) = f(m',m')$, we must also have $\psi(m,m) = \psi(m',m')$. By making precise two intuitions, we will show that this premise cannot in general imply this conclusion. First, every $\psi(m+1,-)$ grows so much faster than $\psi(m,-)$ that $m < m'$ suffices to get $\psi(m,m) < \psi(m',m')$. That is to say, A is strictly monotonic. The conclusion thus rarely ever holds. Second, given some m and limited only by the constraint $m' > m$, there are so many unary primitive recursive functions that we can easily find some m' such that $f(m,m) = f(m',m')$. The premise is thus easy to satisfy.

***A* is strictly monotonic** Laying out the values of ψ in a two-dimensional grid, open-ended on the right and bottom, with $\psi(m,n)$ in the m-th row and n-th column, we get the following impression:

$$\begin{array}{ccccccc}
\psi(0,0)=1 & \psi(0,1)=2 & \psi(0,2)=3 & \psi(0,3)=4 & \psi(0,4)=5 & \psi(0,5)=6 & \cdots \\
\psi(1,0)=2 & \psi(1,1)=3 & \psi(1,2)=4 & \psi(1,3)=5 & \psi(1,4)=6 & \psi(1,5)=7 & \cdots \\
\psi(2,0)=3 & \psi(2,1)=5 & \psi(2,2)=7 & \psi(2,3)=9 & \psi(2,4)=11 & \psi(2,5)=13 & \cdots \\
\psi(3,0)=5 & \psi(3,1)=13 & \psi(3,2)=29 & \psi(3,3)=61 & \psi(3,4)=125 & \psi(3,5)=253 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

Every row is strictly increasing, i.e. $\psi(m,-)$ is strictly monotonic for all m. Every next row begins with the second value of the previous row and grows at least as quickly as that previous row. Therefore, every column is also increasing, i.e. $\psi(-,n)$ is strictly monotonic for all n.

These impressions are made precise in a series of technical lemmas, all of which use nested induction.[6] If the impressions are true, we will easily be able to conclude that $\psi(m,m) < \psi(m',m')$ if $m < m'$.

Lemma 22 (ψ is strictly increasing)**.** *For all m, n, we have $\psi(m,n) > n$.*

Proof. We prove that $A := \{m \in \mathbb{N} | \forall n : \psi(m,n) > n\}$ equals $\mathbb{N}$ inductively, i.e. we prove A contains 0 and is closed under the successor operator.

$0 \in A$ Fix an arbitrary n. $\psi(0,n) = n+1 > n$.

$m \in A$ **implies** $m+1 \in A$ We prove $\forall n : \psi(m+1,n) > n$ by a nested induction, i.e. we prove that $B := \{n \in \mathbb{N} | \psi(m+1,n) > n\}$ equals $\mathbb{N}$.

$0 \in B$ $\psi(m+1,0) = \psi(m,1)$, which is greater than 1 by the outer induction hypothesis ($m \in A$) and thus also greater than 0.

$n \in B$ **implies** $n+1 \in B$ $\psi(m+1,n+1) = \psi(m,\psi(m+1,n)) > \psi(m+1,n) > n$, where the first inequality holds by the outer induction hypothesis and the second by the inner induction hypothesis ($n \in B$).

□

We use the lemma to prove that the rows of ψ are strictly increasing.

Lemma 23 (Every row $\psi(m,-)$ is monotonic)**.** *For all m, n and $n' < n$ we have $\psi(m,n') < \psi(m,n)$.*

Proof. We prove that $A := \{m \in \mathbb{N} | \forall n : \forall n' < n : \psi(m,n') < \psi(m,n)\}$ equals $\mathbb{N}$ inductively.

0 $\in$**A** For any $n' < n$ we have $\psi(0,n') = n'+1 < n+1 = \psi(0,n)$.

$m \in A$ **implies** $m+1 \in A$ We prove that $B := \{n \in \mathbb{N} | \forall n' < n : \psi(m+1,n') < \psi(m+1,n)\}$ equals $\mathbb{N}$ inductively. The case for $n = 1$ is proved separately. This does, of course, not change the validity of the induction.

$0 \in B$ This is vacuously true as there are no $n' < 0$.

[6]As stated before, these lemmas are often used to prove the Majorization lemma of the non-diagonal proof of ψ's non-primitive recursivity.

$1 \in B$ We must prove $\psi(m+1,0) < \psi(m+1,1)$. This is the case because $\psi(m+1,1) = \psi(m,\psi(m+1,0)) > \psi(m+1,0)$ by Lemma 22.

$n+1 \in B$ **implies** $n+2 \in B$ We assume that for all $n' < n+1$, $\psi(m+1,n') < \psi(m+1,n+1)$ and must now prove that for all $n' < n+2$, $\psi(m+1,n') < \psi(m+1,n+2)$. Clearly, it is sufficient to prove that $\psi(m+1,n+1) < \psi(m+1,n+2)$. We see that $\psi(m+1,n+1) = \psi(m,\psi(m+1,n)) < \psi(m,\psi(m+1,n+1)) = \psi(m+1,n+2)$. The inequality holds because $\psi(m+1,n) < \psi(m+1,n+1)$ (since $n+1 \in B$) and $m \in A$.

□

Before we can use this lemma to prove that the columns of ψ are also strictly increasing, we first need a small helper lemma:

Lemma 24. *For all n, $\psi(1,n) = n+2$.*

Proof. It holds for $n=0$ since $\psi(1,0) = \psi(0,1) = 2$. If $\psi(1,n) = n+2$ then $\psi(1,n+1) = \psi(0,\psi(1,n)) = \psi(0,n+2) = n+3$. □

We now combine this lemma with Lemma 23 to prove that the columns of ψ are strictly increasing

Lemma 25 (Every column $\psi(-,n)$ is monotonic)**.** *For all m,n, and $m' < m$ we have $\psi(m',n) < \psi(m,n)$.*

Proof. We prove that $A := \{m \in \mathbb{N} | \forall m' < m : \forall n : \psi(m',n) < \psi(m,n)\}$ equals $\mathbb{N}$ inductively.

$0 \in A$ Vacuously true since there are no $m' < 0$.

$1 \in A$ We only need that for all n, $\psi(0,n) < \psi(1,n)$. This is the case since $\psi(0,n) = n+1$ and $\psi(1,n) = n+2$ by the previous lemma.

$m+1 \in A$ **implies** $m+2 \in A$ We prove that $B := \{n \in \mathbb{N} | \forall m' < m+2 : \psi(m',n) < \psi(m+2,n)\}$ equals $\mathbb{N}$ inductively.[7]

$0 \in B$ It suffices to prove that $\psi(m+1,0) < \psi(m+2,0)$. We see that $\psi(m+1,0) = \psi(m,1) < \psi(m+1,1) = \psi(m+2,0)$, where the inequality holds since $m+1 \in A$.

$n \in B$ **implies** $n+1 \in B$ Since we assume $m+1 \in A$, it suffices to prove that $\psi(m+1,n+1) < \psi(m+2,n+1)$. We see that $\psi(m+2,n+1) = \psi(m+1,\psi(m+2,n))$. Meanwhile $\psi(m+1,n+1) = \psi(m,\psi(m+1,n)) < \psi(m,\psi(m+2,n)) < \psi(m+1,\psi(m+2,n))$. The first inequality holds because $\psi(m+1,n) < \psi(m+2n)$ (since $n \in B$) and by Lemma 23. The second inequality holds because $m+1 \in A$.

□

The conclusion then follows very quickly:

Corollary 26 (A is strictly monotonic)**.** *For all $m < m'$ we have $A(m) = \psi(m,m) < \psi(m',m') = A(m')$. In other words, A is strictly monotonic.*

Proof. This follows by transitivity of $<$ via either one of two routes. We see that $\psi(m,m) < \psi(m,m'), \psi(m',m)$ by Lemma 23 and Lemma 25, respectively. Further, $\psi(m,m'), \psi(m',m) < \psi(m',m')$ by Lemma 25 and Lemma 23, respectively. □

[7]The order of quantification over m' and n has been switched around relative to that in A. This has no effect since both are universal.

We can find $f(m', m') = f(m, m)$ In order to show that Yanofsky's GCT cannot yield a g equal to A, we must show that there are some m, m' with equal inputs to α (i.e. $f(m, m) = f(m', m')$) and unequal expected outputs, i.e. $A(m) = \psi(m, m) \neq \psi(m', m') = A(m')$. We just saw how $m < m'$ leads to $\psi(m, m) < \psi(m', m')$. It is thus sufficient to find two values $m \neq m'$ for which $f(m, m) = f(m', m')$. We show that for every m, we can find infinitely many such m'.

Proposition 27. *Suppose we apply Yanofsky's GCT in* **Set** *with our standard, primitive universal* $f(m, n)$. *Then every* $\alpha : \mathbb{N} \to \mathbb{N}$ *causes* $g = \alpha \circ f \circ \Delta : \mathbb{N} \to \mathbb{N}$ *to differ from* $A : \mathbb{N} \to \mathbb{N}$ *on infinitely many input values.*

Proof. Fix an arbitrary $\alpha : \mathbb{N} \to \mathbb{N}$ and $m \in \mathbb{N}$. Suppose that there is some $m \in \mathbb{N}$ on which g agrees with A: $g(m) = \alpha(f(m, m)) = A(m) = \psi(m, m)$. (If there is no such m, we are done.) There are infinitely many unary primitive recursive programs $P^p_{m'}$ that, on any given input, output the constant value $f(m, m)$. Thus there are infinitely many m' for which $f(m', m') = f(m, m)$. Of course at most one of these infinitely many m' can be equal to m. For the infinitely many other m', we see that $g(m') = \alpha(f(m', m')) = \alpha(f(m, m)) = \psi(m, m)$. By Corollary (26) we know that this is different from $A(m') = \psi(m', m')$. $\square$

We see that this proof depends on the fact that every unary primitive recursive function appears infinitely many times in the enumeration $f(0, -), f(1, -), \ldots$. Although we will not elaborate on it here, it is a priori conceivable that there is e.g. an injective enumeration $\chi_0, \chi_1, \ldots$ where every unary primitive recursive function appears exactly once. We can generalize our findings slightly to applications of Yanofsky's GCT to an altered f that uses any arbitrary enumeration. We also realize that strict monotonicity is not required: we can do with A being almost injective.

Theorem 28. *Assume any enumeration* $\chi_0, \chi_1, \ldots$ *of all unary primitive recursive functions. Suppose we apply Yanofsky's GCT in* **Set** *with* $f'(m, n) := \chi_m(n)$. *(We see that any unary function is representable in* f' *iff it is primitive recursive.) Suppose that* $A' : \mathbb{N} \to \mathbb{N}$ *is such that for each* n*, there are at most finitely many* m *with* $A(m) = n$. *Then every* $\alpha : \mathbb{N} \to \mathbb{N}$ *causes* $g = \alpha \circ f' \circ \Delta : \mathbb{N} \to \mathbb{N}$ *to differ from* A' *on infinitely many input values.*

Proof. Fix an arbitrary α and m such that $g(m) = \alpha(\chi_m(m)) = A'(m)$. Suppose, without loss of generality, that $m \neq 0$. Consider the countably infinitely many unary primitive recursive functions $(h_i)_{i \in \mathbb{N}}$ which map 0 to i and all other inputs to the constant value $\chi_m(m)$. Every such h_i has at least one m_i such that $h_i = \chi_{m_i}$. All m_i are different so at most one can be 0 and we have infinitely many m_i left for which $h_i(m_i) = \chi_m(m)$ and thus $\chi_{m_i}(m_i) = \chi_m(m)$. This causes $\alpha(\chi_{m_i}(m_i)) = \alpha(\chi_m(m)) = A'(m)$ for infinitely many m_i. Since A' is almost injective, there are infinitely many m_i for which $g(m_i) = \alpha(\chi_{m_i}(m_i)) = \alpha(\chi_m(m)) = A'(m)$ must be different from $A'(m_i)$. $\square$

We conclude that there is a big class of unary functions for which no straightforward application of Yanofsky's GCT allows us to conclude that it is not primitive recursive. Note that we have not proven that it is impossible to apply Yanofsky in another way. We have, for example, only examined one straightforward definition of f that ensures representability is equivalent to primitive recursivity. We suspect, however, that other definitions of f will be sufficiently similar that Yanofsky cannot be applied in any way.

4 Recursion theorem

In the previous section we applied Yanofsky's GCT to primitive recursive functions. In this section we turn our attention to another area of computability theory. We will investigate whether we can prove a specific existing theorem using the GCT.

4.1 Prerequisites

Definition 29. *A set X is* effectively denumerable *iff there is a bijection $\imath$ between X and $\mathbb{N}$ where both $\imath$ and $\imath^{-1}$ can informally be said to be effectively computable by some kind of machine or systematic process. Note that this must necessarily be in an informal sense, since computability is only defined for functions of signature $\mathbb{N}^k \to \mathbb{N}$ for some k.*

Proposition 30. *The set $\mathscr{I}$ of URM instructions is effectively denumerable.*

For a full proof, see [Cut80].

Proof sketch. There are four kinds of instructions. We construct helper bijections $\pi : \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}$ and $\zeta : \mathbb{N}^+ \times \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}$. We then realize four bijections, each between one kind of instruction and the natural numbers, and zip them together into one effectively computable bijection β as follows: $\beta(Z(n)) := 4(n-1)$, $\beta(S(n)) := 4(n-1)+1$, $\beta(T(m,n)) := 4(\pi(m,n))+2$, $\beta(J(m,n,q)) := 4\zeta(m,n,q) + 3$. The inverse β^{-1} is clearly also effectively computable: we first determine the kind of the instruction using the remainder after division by 4 and then use the bijection of that particular kind to obtain an instruction. $\square$

Proposition 31. *The set $\mathscr{P}$ of all URM programs is effectively denumerable.*

For a full proof, see [Cut80].

Proof sketch. We construct a helper bijection $\tau : \mathbb{N}^* \to \mathbb{N}$ between finite sequences of natural numbers and the natural numbers themselves. It is an effective denumeration of $\mathbb{N}$, i.e. τ and τ^{-1} are effectively computable. The function $\gamma : \mathscr{P} \to \mathbb{N} : (I_1, \ldots, I_k) \mapsto \tau(\beta(I_1), \ldots, \beta(I_k))$ is then a bijection. Both γ and γ^{-1} are effectively computable, making γ an effective denumeration of $\mathscr{P}$. $\square$

We will keep referring to γ outside of the proof. γ^{-1} maps every natural number to a program. Since γ is a bijection, γ^{-1} is a surjection (and hence an enumeration of $\mathscr{P}$) and an injection (i.e. the enumeration has no repetitions).

Definition 32. *Let P_n be the n-th program as enumerated by γ^{-1}, i.e. $P_n := \gamma^{-1}(n)$. n is called the* index *of P_n.*

As we have seen before, every URM program computes a partial function: it takes k inputs in the first k registers and halts with an output in register R_0. A technical point is to be made: we can supply any one program with different lengths of input. Even if the registers actually used by the program are not at all the same as those containing the input, this is technically irrelevant as *some* function is computed. Through the enumeration $(P_n)_{n\in\mathbb{N}}$ of $\mathscr{P}$, we can thus enumerate all functions of any one certain arity:

Definition 33. *Let $\phi^m_n : \mathbb{N}^m \rightharpoonup \mathbb{N}$ be the unique m-ary partial recursive function computed by P_n.*

Proposition 34. *Let $\mathscr{C}^m$ denote the set of m-ary partial recursive functions. The function $\phi^m : \mathbb{N} \to \mathscr{C}^m : n \mapsto \phi^m_n$, more often referred to as a sequence $(\phi^m_n)_{n\in\mathbb{N}}$, is an enumeration. It has repetitions, i.e. it is not injective. In fact, every m-ary partial recursive function appears countably infinitely many times in the enumeration.*

Proof. Fix an arbitrary computable function $f : \mathbb{N}^m \rightharpoonup \mathbb{N}$. By definition, there must be a program P computing f. If we define $n := \gamma(P)$ we see that $P = \gamma^{-1}(\gamma(P)) = P_n$. Then $\phi_n^m = f$ since P_n computes both f and ϕ_n^m and any program only computes one function of each arity.

In fact, it is not difficult to see that there are countably infinitely many programs $P_{n_0}, P_{n_1}, \ldots$ computing f.[8] This means that $f = \phi_{n_0}^m = \phi_{n_1}^m$. □

We will often leave out the superscript in ϕ_n^m and write ϕ_n if the arity is clear from context.

Proposition 35 (s-m-n theorem)**.** *Suppose $m, n \geq 1$ are natural numbers. Then there exists a total computable function $s_n^m : \mathbb{N}^{m+1} \to \mathbb{N}$ such that for all $e \in \mathbb{N}$ and $\mathbf{x} \in \mathbb{N}^m$, $\phi_e^{m+n}(\mathbf{x}, -) = \phi^n_{s_n^m(e,\mathbf{x})}$.*

Specifically, for every partial computable function $f : \mathbb{N}^2 \rightharpoonup \mathbb{N}$ there is some total computable $s : \mathbb{N} \to \mathbb{N}$ such that for all $x \in N$, $f(x, -) = \phi_{s(x)}$.

For a full proof, see [Cut80]. We want a computable function s_n^m that, given a program index e and an input tuple $\mathbf{x}$, produces the index of a program computing the n-ary function that maps $\mathbf{y}$ to $\phi_e^{m+n}(\mathbf{x}, \mathbf{y})$. We can think of this program as P_e except that the first m input variables are hard-coded to $\mathbf{x}$. By the Church-Turing thesis, we need only provide a *systematic procedure* to obtain from e and $\mathbf{x}$ an index of such a hard-coded program. Since $\mathscr{P}$ is effectively denumerable with γ^{-1}, the systematic procedure need only informally produce the program itself.

Proof sketch. Suppose we are given e and $\mathbf{x} \in \mathbb{N}^m$. We can think of instructions that move the contents of the first n registers m places to the right: the value of R_1 ends up in R_{1+m}, etc. up to that of R_n ending up in R_{m+n}. We can also construct instructions setting the contents of the first m registers to the respective components of $\mathbf{x}$. We let these instructions be followed by those of the program P_e.

All preceding instructions depend on m, n, e and $\mathbf{x}$. We define s_n^m as the composition of the above construction with the effectively computable γ. This way, we obtain an s_n^m which, upon receiving $(e, \mathbf{x})$ as input, computes the instructions of a program whose function maps $\mathbf{y}$ to $\phi_{\mathbf{e}}(\mathbf{x}, \mathbf{y})$. After computing this program it outputs its index. □

4.2 The recursion theorem

In this section we refer to a deep result in computability theory. It has been put forward in some form by Kleene in 1938,[Kle38] but we will be using Rogers' formalization.[Rog67] We will hereafter prefer non-attribution over misattribution and refer to it simply as *the recursion theorem.*

Theorem 36 (Recursion theorem)**.** *Let $F : \mathbb{N} \to \mathbb{N}$ be a total computable function. Then there exists a number $n_0 \in \mathbb{N}$ such that $\phi_{F(n_0)} = \phi_{n_0}$.*

We now provide a standard proof for the recursion theorem. See, for example, [Cut80]. Soon we will investigate other ways to prove the same theorem.

Proof of Theorem 36. We observe that the function $(x, y) \mapsto \phi_{F(\phi_x(x))}(y)$ is partial computable.[9] By the s-m-n theorem, therefore, there is a total computable $s : \mathbb{N} \to \mathbb{N}$ such that for all inputs x,

$$\phi_{F(\phi_x(x))}(y) \simeq \phi_{s(x)}(y). \tag{2}$$

[8]One URM program can be trivially modified in infinitely many ways without changing its function. For example, if the program consists of k instructions one can always append a $J(0, 0, k + 2)$ instruction.

[9]It is not total because ϕ_x might be undefined at x or $\phi_{F(\phi_x(x))}$ might be undefined at y.

Since this s is computable, it is equal to some ϕ_m. Evaluating Eq. (2) at $x = m$ and realizing that $m \in \phi_m$, we see that $\phi_{F(\phi_m(m))} = \phi_{\phi_m(m)}$. Therefore we can choose $\phi_m(m)$ as our number n_0 for which $\phi_{F(n_0)} = \phi_{n_0}$. $\square$

The recursion theorem leads to some unexpected results.

Example 37 (Quines)**.** *Programming languages are a way of associating (a subset of) the finite strings over some alphabet with computable functions. Most programming languages only accept a strict subset of all possible strings; these are called the programs. There are countably infinitely many programs, all of which describe one computable function. Every computable function is described by infinitely many programs.*

We can thus regard $P_0, P_1, \ldots$ or, equivalently, their numbers $0, 1, \ldots$, as programs in a programming language.

Computer scientists have long been interested in self-reproducing programs*,[BM72] also called* quines*. By this we mean programs that, whatever their input they are given, output their own source code. In our context, where the programming language is the set of natural numbers, this means quines output their own index.*

The recursion theorem tells us that our programming language has quines, i.e. there is some n such that $\phi_n(y) = n$ for all y.

We see that $G(x, y) := x$ describes a partial recursive function. By the s-m-n theorem, there exists a total computable function s such that for all x and y, $\phi_{s(x)}(y) = G(x, y) = x$. Applying the recursion theorem to s we get some n_0 such that $\phi_{s(n_0)} = \phi_{n_0}$. We see that $\phi_{n_0}(y) = \phi_{s(n_0)}(y) = G(n_0, y) = n_0$ for any input y.

4.3 The recursion theorem via Cantor: Yanofsky's attempt

We discern some diagonal character in the proof of Theorem 36. It hinges upon the application of a function ϕ_x to its own number x. It seems reasonable to suspect that this proof can be shown to be an instance of Yanofsky's GCT. In a 2003 paper, Yanofsky claimed to have done just that.[Yan03] More accurately, they framed the recursion theorem as an instance of Corollary (12). We paraphrase their argumentation here.

Yanofsky's argument Yanofsky works in a set-theoretic setting and puts forward the following commutative diagram:

$$\begin{tikzcd}
\mathbb{N} \times \mathbb{N} \arrow[r, "f:(m,n)\mapsto\phi_{\phi_m(n)}"] & \mathscr{C}^1 \arrow[d, "\alpha:\phi_x\mapsto\phi_{F(x)}"] \\
\mathbb{N} \arrow[u, "\Delta"] \arrow[r, "g"'] & \mathscr{C}^1
\end{tikzcd}$$

Yanofsky claims that $g(x) = \phi_{F(\phi_x(x))}$ for all x and notes that the function $(x, y) \mapsto g(x)(y) = \phi_{F(\phi_x(x))}(y)$ is partial computable. By the s-m-n theorem, there would then be some total ϕ_m such that for all n, $g(n) = \phi_{F(\phi_n(n))} = \phi_{\phi_m(n)} = f(m, n)$. Thus g would be represented in f by this m. Yanofsky concludes from this that α has a fixpoint, using the contrapositive corollary of the GCT.

We see multiple problems with this, however. First of all, we do not know in which category this diagram lives. It cannot be **Set** since f is a partial function. As we will see, the solution is not as simple as taking the category of sets and *partial* functions.

Second, α is not guaranteed to be a well-defined function. α operates on the level of partial computable functions. It must map equal partial computable functions $\phi_{n_1} = \phi_{n_2}$ with different indices $n_1 \neq n_2$ to equal functions and thus $\phi_{F(n_1)} = \phi_{F(n_2)}$. It is not guaranteed that F satisfies this constraint as each partial computable function has infinitely many indices. In fact, we can find a very simple counterexample:

Example 38. *Let $F(n) := n + 1$. This is clearly a total computable function. Assume $\alpha : \phi_x \mapsto \phi_{F(x)}$ is a well-defined function. Let $n_1 < n_2$ be a colliding pair, i.e. $\phi_{n_1} = \phi_{n_2}$. Such n_1, n_2 exist since every partial computable function has infinitely many indices in the enumeration $(\phi_n)_{n\in\mathbb{N}}$. Then $\phi_{n_1+1} = \phi_{F(n_1)} = \alpha(\phi_{n_1}) = \alpha(\phi_{n_2}) = \phi_{F(n_2)} = \phi_{n_2+1}$. In fact, it is not difficult to see that by induction, $\phi_{n_1+k} = \phi_{n_2+k}$ for every k. This means that starting from index n_1, the enumeration $(\phi_n)_{n\in\mathbb{N}}$ keeps enumerating the same functions in a cycle of length at most $n_2 - n_1$. The surjectivity of the enumeration contradicts the fact that there are infinitely many partial computable functions.*

A third problem with Yanofsky's argument is that the s-m-n theorem does not yield such a strong result. Regardless of α being well-defined, the function $(x, y) \mapsto \phi_{F(\phi_x(x))}(y)$ is well-defined and partial computable. The s-m-n theorem only guarantees us the existence of some total ϕ_m such that we have $\phi_{F(\phi_x(x))}(y) \simeq \phi_{\phi_m(x)}(y)$ for all x, y. This does not yield that $\phi_{F(\phi_x(x))} = \phi_{\phi_m(x)}$ for all x, since there are $x \notin \operatorname{dom} \phi_x$ for which $\phi_{F(\phi_x(x))}$ thus is undefined while $\phi_{\phi_m(x)}$ is always defined.

In the rest of Section 4 we will investigate different ways to tackle these problems.

4.4 Friedberg numberings

To make Yanofsky's argument work, we must alter some aspect to make α well-defined. We just saw in Example 38 that the existence of any pair of colliding indices $n_1 \neq n_2$ with $\phi_{n_1} = \phi_{n_2}$ already precludes this for $F(n) := n + 1$. We thus wonder what happens under alternative enumerations without collisions. Such enumerations take the shape of $(\phi_{h(0)}, \phi_{h(1)}, \ldots)$ for some function h. An idea by Richard Friedberg led to such a numbering, now called a *Friedberg numbering.* [Cut80; Fri58]

Proposition 39 (Friedberg numbering)**.** *For every arity m, there is a total computable function $h^m : \mathbb{N} \to \mathbb{N}$ such that $(\phi^m_{h^m(n)})_{n\in\mathbb{N}}$ is an enumeration of the m-ary partial recursive functions without repetitions.*

For a proof, see [Kum90]. We will mostly be concerned with h^1 and leave out arity indications. We will also refer to the Friedberg numbering $(\phi_{h(n)})_{n\in\mathbb{N}}$ as $\phi \circ h$. Like the original numbering ϕ, the Friedberg numbering $\phi \circ h$ is a surjection from $\mathbb{N}$ onto $\mathscr{C}^1$. Unlike the original numbering, $\phi \circ h$ is also injective and hence a bijection from $\mathbb{N}$ to $\mathscr{C}^1$. There are no colliding indices. Note that h itself is injective but clearly not surjective.

Yanofsky's GCT is not able to prove a version of the recursion theorem for the Friedberg numbering, however. This is because the recursion theorem does not even hold for the Friedberg numbering:

Proposition 40 (No recursion theorem for Friedberg numbering)**.** *It is not the case that for every total computable function $F : \mathbb{N} \to \mathbb{N}$ there exists an $n_0 \in \mathbb{N}$ such that $\phi_{h(F(n_0))} = \phi_{h(n_0)}$.*

Proof. By contradiction. Suppose that there would be such an n_0 for every such F. Then since $\phi \circ h$ is injective, every F would have an n_0 such that $F(n_0) = n_0$, i.e., every total computable function would have a fixpoint. This is clearly not the case, since $F(n) := n + 1$ describes a total computable function without fixpoints. $\square$

From Example 38 we could conclude that $\alpha(\phi'_n) := \phi'_{F(n)}$ cannot describe a well-defined function for any surjective numbering ϕ' with at least one collision.

We now conclude that although $\alpha(\phi'_n) := \phi'_{F(n)}$ *does* describe a well-defined function when ϕ' is a bijective numbering, the recursion theorem does not hold for it anyway. We will therefore not be able to prove it using Yanofsky's GCT or any other method.

4.5 Categories that allow for some partiality

In the remaining sections of Section 4 we attempt to solve the problems with Yanofsky's argument using a different approach. We leave the numbering ϕ untouched. We will instead interpret Yanofsky's definitions in categories that relax the totality and even functionality constraints for morphisms.

Let us now explore some relaxations such that Yanofsky's definition of f, a partial function, becomes a valid morphism. In this stage we are reluctant to simply allow all partial functions. (We do consider this option in Section 4.6.)

We first propose a category where the objects are sets and $\mathrm{Hom}(A, B) := \{f : A \rightharpoonup B \mid \operatorname{dom} f \neq \emptyset\}$, i.e. all partial functions are allowed except the nowhere-defined function. Composition is the ordinary function composition. We immediately see that these homomorphism classes are not closed under composition. Consider this counterexample:

$$\begin{tikzcd}
2 \arrow[r, "f"] & 2 \arrow[r, "g"] & 2 \\
0 \arrow[r, mapsto] & 0 & 0 \\
1 & 1 \arrow[r, mapsto] & 1
\end{tikzcd}$$

Both f and g are somewhere-defined but their composition $g \circ f$ is nowhere-defined and therefore not a valid morphism. Our example is not a category.

The requirement of being somewhere-defined turns out to be insufficient. In the preceding counterexample we were free to choose g such that its domain had no overlap with the range $\operatorname{ran} f$ of f, where $\operatorname{ran} f := \{b \in B \mid \exists a \in A : f(a) = b\}$. Our second proposal is a category where functions are required to have a much larger domain: $\mathrm{Hom}(A, B) := \{f : A \rightharpoonup B \mid |\operatorname{dom} f| = |A|\}$. Yanofsky's function $\mathbb{N} \times \mathbb{N} \overset{f}{\rightharpoonup} \mathscr{C}^1$ is defined on (countably) infinitely many input values: $|\operatorname{dom} f| = \omega = |A|$. It is therefore a valid morphism. However, we again manage to find a composition of two morphisms that is not a morphism:

$$\begin{tikzcd}
\mathbb{N} \arrow[r, "f"] & \mathbb{N} \arrow[r, "g"] & \mathbb{N} \\
0 \arrow[r, mapsto] & 0 & 0 \\
1 \arrow[ur, mapsto] & 1 \arrow[ur, mapsto] & \\
2 \arrow[uur, mapsto] & 2 \arrow[uur, mapsto] & \\
3 \arrow[uuur, mapsto] & 3 \arrow[uuur, mapsto] & \\
4 \arrow[uuuur, mapsto] & 4 \arrow[uuuur, mapsto] & \\
\vdots & \vdots &
\end{tikzcd}$$

Note that $|\operatorname{dom} g| = |\mathbb{N} \setminus \{0\}| = |\mathbb{N}| - 1 = |\mathbb{N}|$, making g a morphism. The composition $g \circ f$, however, is again nowhere-defined and therefore not a valid morphism. Again our example is not a category.

We see that in the preceding counterexample we were able to compose two functions into a nowhere-defined function where the former component maps all its inputs to the one element on which the latter component is not defined. We now try to rule out this possibility by restricting the morphism classes. $\mathrm{Hom}(A, B) := \{f : A \rightharpoonup B \,|\, |\mathrm{ran}\, f| \geq \min\{|A|, |B|\}\}$. The range of f is now required to be as big as B except if A is too small for this to be possible. In the latter case, the range must simply be at least as big as A.[10] We can again find a counterexample against this:

$$\begin{tikzcd}
\mathbb{N} \arrow[r, harpoon, "f"] & \mathbb{N} \arrow[r, harpoon, "g"] & \mathbb{N} \\
0 \arrow[r, mapsto] & 0 & 0 \\
1 & 1 \arrow[r, mapsto] & 1 \\
2 \arrow[r, mapsto] & 2 & 2 \\
3 & 3 \arrow[r, mapsto] & 3 \\
4 \arrow[r, mapsto] & 4 & 4 \\
5 & 5 \arrow[r, mapsto] & 5 \\
\vdots & \vdots & \vdots
\end{tikzcd}$$

We see that $|\mathrm{ran}\, f|$ and $|\mathrm{ran}\, g|$ are equal to $|\mathbb{N}| = \min\{|\mathbb{N}|, |\mathbb{N}|\}$. The composition $g \circ f$ is not a morphism since its (empty) range is not greater than $\mathbb{N}$.

In a final attempt to include Yanofsky's f in a class of morphisms that is closed under composition we decide to broaden the morphism class again. $\mathrm{Hom}(A, B) := \{f : A \rightharpoonup B \,|\, |\mathrm{ran}\, f| \geq \min\{|\mathrm{dom}\, f|, |B|\}\}$. This makes Yanofsky's f a valid morphism. The composition $g \circ f$ (the empty function) of the previous example is valid under the new criterion: $\mathrm{ran}(g \circ f) = \mathrm{dom}(g \circ f) = \emptyset$. However, we can slightly modify the example to see that, again, this class is not closed under composition. We can produce the following infinite example:

$$\begin{tikzcd}
\mathbb{N} \arrow[r, harpoon, "f"] & \mathbb{N} \arrow[r, harpoon, "g"] & \mathbb{N} \\
0 \arrow[r, mapsto] & 0 \arrow[r, mapsto] & 0 \\
1 & 1 \arrow[r, mapsto] & 1 \\
2 \arrow[r, mapsto] & 2 \arrow[ruu, mapsto] & 2 \\
3 & 3 \arrow[r, mapsto] & 3 \\
4 \arrow[r, mapsto] & 4 \arrow[ruuuu, mapsto] & 4 \\
5 & 5 \arrow[r, mapsto] & 5 \\
\vdots & \vdots & \vdots
\end{tikzcd}$$

Here, $g \circ f : \mathbb{N} \rightharpoonup \mathbb{N}$ is the function that maps all even numbers to 0. Its range, the singleton $\{0\}$, is clearly strictly smaller than $\mathbb{N}$ and its domain $\mathrm{dom}(g \circ f)$, the set of even numbers. We

[10] Note that Yanofsky's $f : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathscr{C}^1$ does indeed satisfy this as it is surjective.

can also come up with a finite variant of the counterexample:

$$\begin{tikzcd}
3 \arrow[r, "f"] & 3 \arrow[r, "g"] & 2 \\
0 \arrow[r, mapsto] & 0 \arrow[r, mapsto] & 0 \\
1 & 1 \arrow[r, mapsto] & 1 \\
2 \arrow[r, mapsto] & 2 \arrow[uur, mapsto] &
\end{tikzcd}$$

Both f and g have a range at least as big as their domain or codomain. However, $g \circ f$ has range of size 1 and a domain and codomain of size 2. Once more we conclude that our class of morphisms is not closed under composition and that, hence, this is not a category.

We started this section with the realization that Yanofsky's $f : \mathbb{N} \times \mathbb{N} \rightharpoonup \mathscr{C}^1$ is not a morphism in **Set** since it is not a total function. We have tried to find a category in which Yanofsky's f *is* a valid morphism. More concretely, we examined various potential categories which admitted many but not all partial functions. None of these proposals turned out to define an actual category, since none of them had morphism classes that were closed under composition. This does not prove that it is impossible to find such a category that admits Yanofsky's f. We do hope, however, that we gave the reader a feeling for the difficulty of this endeavour and do not investigate it further.

4.6 Category of partial functions

After trying and failing to find a category that admitted Yanofsky's f but not all partial functions it is time for a more sure-fire approach. In this section we will investigate the category of sets with all partial functions between them. Before we do so, however, let us examine in greater detail how (the contrapositive corollary of) the generalized Cantor theorem worked in practice so far. This is necessary because we will be confronted with categories that do not have elements.

In our applications of the GCT, we were given sets T, Y and functions $f : T \times T \to Y, \alpha : Y \to Y$. This amounted to the objects and morphisms $T \times T \xrightarrow{f} Y \xrightarrow{\alpha} Y$ in **Set**. We provided some infrastructure independent of f, α. First, we provided a morphism $T \xrightarrow{\Delta} T \times T$ and precomposed it to $\alpha \circ f$. We also provided an object E and a function $e : T \to \mathrm{Hom}_{\mathbf{Set}}(E, T)$ that somehow "embedded" elements of T into the set of morphisms from E to T. In **Set**, E was any singleton set $\{*\}$ and e was the bijection described by $e(t) := (* \mapsto t)$. Last, we also provided a morphism $T \xrightarrow{\imath} E \times T$. In **Set**, $\imath$ was the unique morphism described in Section 2.3, namely $\imath(t) := (*, t)$.

We started by proving that

$$\alpha \circ f \circ \Delta = f \circ (e(t) \times \mathrm{Id}_T) \circ \imath \tag{3}$$

held in the category. From this we drew the conclusion that for any element t of T,

$$\alpha \circ f \circ \Delta \circ e(t) = f \circ (e(t) \times \mathrm{Id}_T) \circ \imath \circ e(t). \tag{4}$$

This step of reasoning is valid in any category. From the equality

$$(e(t) \times \mathrm{Id}_T) \circ \imath \circ e(t) = \Delta \circ e(t) \tag{5}$$

we obtained

$$\alpha \circ f \circ \Delta \circ e(t) = f \circ \Delta \circ e(t), \tag{6}$$

from which we concluded that $f \circ \Delta \circ e(t)$ somehow represents a fixpoint of the function α. In **Set**, which has elements, this conclusion was obvious. $f \circ \Delta \circ e(t)$, as a function from $E = \{*\} = 1$

to Y, bijectively corresponded with $f(\Delta(t)) = f(t,t)$, which then in fact *was* a fixpoint of the function α.

We wish to have a clear and rigourous framework to discuss partial functions. Let **Set**$*$ be the category of pointed sets. That is, a member of ob **Set**$*$ is a set A together with a distinguished element $\perp_A \notin A$. We will often leave out the subscript in $\perp_A$ and write this pair $(A, \perp_A)$ as $A^\perp$. $\mathrm{Hom}_{\mathbf{Set}*}(A^\perp, B^\perp)$ is then the set of all total functions from $A \uplus \{\perp_A\}$ to $B \uplus \{\perp_B\}$ that map $\perp$ to $\perp$. Note that morphisms are allowed to map $a \in A$ to $\perp$ as well. The set of partial functions $A \rightharpoonup B$ is in bijective correspondence with $\mathrm{Hom}_{\mathbf{Set}*}(A^\perp, B^\perp)$: mapping $a \in A$ to $\perp$ is the equivalent of being undefined on input a. We let categorical composition be function composition, which is associative and under which our class of homomorphisms is closed. The identity morphisms are of course the identity functions. Note that **Set**$*$ has products, namely the cartesian product: $(A, \perp_A) \times (B, \perp_B)$ is $((A \otimes B) \cup (\{\perp_A\} \otimes B) \cup (A \otimes \{\perp_B\}), (\perp_A, \perp_B))$ where $\otimes$ denotes cartesian, set-theoretical product.

We now attempt to let the GCT go through in **Set**$*$. This might let us prove the recursion theorem using the GCT while avoiding the problems mentioned in Section 4.3, particularly the partiality problem. Suppose we are given sets T, Y and partial functions $f : T \times T \rightharpoonup Y$ and $\alpha : Y \rightharpoonup Y$. We define the morphism $T^\perp \times T^\perp \xrightarrow{f} Y^\perp$ to be a function that maps (t,s) to $f(t,s)$ if that is defined and to $\perp$ if it is not. Of course it maps $(\perp, \perp)$ to $\perp$. It is not important what the other inputs are mapped to. We let the morphism $Y^\perp \xrightarrow{\alpha} Y^\perp$ be the function that maps y to $\alpha(y)$ if that is defined and to $\perp$ if it is not. The diagonal function $\Delta : T \to T \times T$ automatically leads to the morphism $T^\perp \xrightarrow{\Delta} T^\perp \times T^\perp$.

Set$*$ does not have elements. Since, in general, $\left|\mathrm{Hom}(A^\perp, B^\perp)\right| = (|B| + 1)^{|A|}$, there is no E such that $\mathrm{Hom}_{\mathbf{Set}*}(E, X)$ is in bijective correspondence with X for every set X. We must content ourselves with an approximation: we let $E = \{*\}^\perp$ be a singleton plus a separate distinguished element and achieve $|\mathrm{Hom}(E, X)| = |X| + 1$. We can then have the embedding $e : T \to \mathrm{Hom}(E, T)$ where $e(t)$ maps $*$ to t (and $\perp$ to $\perp$). Finally, we let the morphism $T \xrightarrow{\imath} E \times T$ be the function that maps $t \in T$ to $(*, t)$ (and $\perp$ to $\perp$).

Following the same steps of reasoning as before, we see that once we have

$$\alpha \circ f \circ \Delta = f \circ (e(t) \times \mathrm{Id}_T) \circ \imath \tag{7}$$

for some t then we can conclude that

$$\alpha \circ f \circ \Delta \circ e(t) = f \circ (e(t) \times \mathrm{Id}_T) \circ \imath \circ e(t). \tag{8}$$

Indeed,

$$(e(t) \times \mathrm{Id}_T) \circ \imath \circ e(t) = \Delta \circ e(t) \tag{9}$$

are equal morphisms from E to $T^\perp \times T^\perp$. As these are valid morphisms they both map $\perp$ to $\perp$ and to check equality we need only investigate their behaviour on $*$. The left-hand side maps $*$ to $((e(t) \times \mathrm{Id}_T) \circ \imath)(t) = (e(t) \times \mathrm{Id}_T)(*, t) = (t,t)$ and the right-hand side maps it to $\Delta(t) = (t,t)$. We would thus conclude

$$\alpha \circ f \circ \Delta \circ e(t) = f \circ \Delta \circ e(t). \tag{10}$$

The left-hand side maps $*$ to $\alpha(f(t,t))$ if f is defined on (t,t) and α is defined on $f(t,t)$ and maps it to $\perp$ otherwise. The right-hand side maps $*$ to $f(t,t)$ if f is defined on (t,t) and maps it to $\perp$ otherwise.

From Eq. (10) we would be able to conclude that for any $t \in T$, if $\alpha(f(t,t))$ is defined then $f \circ \Delta \circ e(t)$ "represents" a fixpoint of α in the sense that $(f \circ \Delta \circ e(t))(*) = f(t,t) = \alpha(f(t,t))$ *is* a fixpoint of the function α. If $\alpha(f(t,t))$ is not defined then Eq. (10) only tells us that $\perp = \perp$, teaching us nothing.

We wish to apply these reasoning steps to the second recursion theorem. This means that we are given a total function $F : \mathbb{N} \to \mathbb{N}$, that $T = \mathbb{N}$, that $Y = \mathscr{C}^1$ (although we will leave out the superscript), that $\mathbb{N}^\perp \times \mathbb{N}^\perp \xrightarrow{f} \mathscr{C}^\perp$ maps (m, n) to $\phi_{\phi_m(n)}$ if it is defined and to $\perp$ otherwise, that $\mathbb{N}^\perp \xrightarrow{\alpha} \mathbb{N}^\perp$ maps ϕ_x to $\phi_{F(x)}$, that $E = \{*\}^\perp$, that $e(m) \in \mathrm{Hom}_{\mathbf{Set}*}(E, \mathbb{N})$ maps $*$ to m and that $\mathbb{N}^\perp \xrightarrow{\imath} E \times \mathbb{N}^\perp$ maps m to $(*, m)$. We solved the first problem we mentioned in Section 4.3 by explicitly allowing partial functions. We are, however, still confronted with the second problem: α is not necessarily a well-defined function. We therefore repeat the reasoning steps of Eq. (7) through Eq. (10) with one alteration: we contract $\alpha \circ f$ into one morphism $\mathbb{N}^\perp \times \mathbb{N}^\perp \xrightarrow{f'} \mathscr{C}^\perp$. It maps (m, n) directly to $\phi_{F(\phi_m(n))}$ if it is defined and to $\perp_\mathscr{C}$ otherwise and it maps $\perp = (\perp_\mathbb{N}, \perp_\mathbb{N})$ to $\perp_\mathscr{C}$. We again do not care about its behaviour on other input values. If we have

$$f' \circ \Delta = f \circ (e(m) \times \mathrm{Id}_\mathbb{N}) \circ \imath \tag{11}$$

for some m then also

$$f' \circ \Delta \circ e(m) = f \circ (e(m) \times \mathrm{Id}_\mathbb{N}) \circ \imath \circ e(m). \tag{12}$$

Since indeed Eq. (9) holds for any $T^\perp$ and thus also for $\mathbb{N}^\perp$ this leads to

$$f' \circ \Delta \circ e(m) = f \circ \Delta \circ e(m). \tag{13}$$

Notice that the left-hand side maps $*$ to $\phi_{F(\phi_m(m))}$ if it is defined and the right hand side maps $*$ to $\phi_{\phi_m(m)}$ if it is defined. This means that if ϕ_m is defined on m then we have the desired conclusion of the recursion theorem: $\phi_{F(\phi_m(m))} = \phi_{\phi_m(m)}$ (since F is total). If ϕ_m is not defined on m, however, we only get the useless equation $\perp = \perp$.

Let us now examine if these reasoning steps are of any use to us. The left-hand side in Eq. (11) maps any $n \in \mathbb{N}$ to $\phi_{F(\phi_n(n))}$ if ϕ_n is defined on n and to $\perp$ otherwise. For any $m \in \mathbb{N}$ the right-hand side maps any $n \in \mathbb{N}$ to $\phi_{\phi_m(n)}$ if ϕ_m is defined on n and to $\perp$ otherwise. To achieve equality, therefore, we need some m such that $\phi_m(n) \simeq \phi_n(n)$ for all $n \in \mathbb{N}$. Yanofsky's argument never produces such an m, however. The ϕ_m which Yanofsky obtains by applying the s-m-n theorem to the partial function $(x, y) \mapsto \phi_{F(\phi_x(x))}(y)$ has the property that $\phi_{\phi_m(x)}(y) \simeq \phi_{F(\phi_x(x))}(y)$ for all x, y. It is also total. Thus $\phi_m(n)$ is always defined while there are many $n \in \mathbb{N}$ for which $\phi_n(n)$ is not (e.g. the nowhere-defined function appears infinitely often in the enumeration), so Eq. (11) cannot hold. Since we know that ϕ_m is defined on m we have $\phi_{\phi_m(m)} = \phi_{F(\phi_m(m))}$, which is the desired conclusion of the recursion theorem. Yanofsky's argument therefore does get us Eq. (13) (or, equivalently, Eq. (12)). In order to see this, however, we had to carry out the whole ordinary proof of the recursion theorem. Our goal of proving the recursion theorem *using* Yanofsky's GCT has not been reached.

Let us now try to find some other m' such that Eq. (11) *does* hold (for m'). In other words, we must find some m' such that $(n \mapsto \phi_{F(\phi_n(n))}) = (n \mapsto \phi_{\phi_{m'}(n)})$. We can use the ϕ_m that Yanofsky obtained from the s-m-n theorem so that $P_{m'}$, given n, first computes $P_n(n)$ then throws it away and computes and outputs $P_m(n)$. This makes sure that $\phi_{m'}(n)$ is not defined if $\phi_n(n)$ is not. Alternatively, we can simply define $\phi_{m'} : n \mapsto F(\phi_n(n))$ without even using the s-m-n theorem. Either way, we see that Eq. (11) holds for m'. From this follow Eq. (12) and Eq. (13). The desired conclusion of the recursion theorem, $\phi_{F(\phi_{m'}(m'))} = \phi_{\phi_{m'}(m')}$, then holds as long as $\phi_{m'}$ is defined on m'. There is no reason why that would be the case, however. We expressly gave up totality when we moved from Yanofsky's ϕ_m to $\phi_{m'}$.

We conclude that this approach in the category **Set**$*$ of partial functions does not readily allow us to prove the recursion theorem. Although it does make f a valid morphism, α still is not. We circumvented this by giving up the structure of the diagram and contracting $\alpha \circ f$ into one morphism f'. Sticking to this we find an m for which indeed $\phi_{F(\phi_m(m))} = \phi_{\phi_m(m)}$. However, this is derived by carrying out the ordinary, non-GCT proof of the recursion theorem. We could only recover the GCT proof steps up to Eq. (12) and could not recover the important premise, Eq. (11). We found a value m' for which the premise Eq. (11) *did* in fact hold but it had to be chosen such that the conclusion Eq. (13) did not prove the recursion theorem.

4.7 Category of relations

In our final attempt to prove the recursion theorem with a technique based on Yanofsky's GCT we make use of a another category, **Rel**. The objects of **Rel** are again all sets. $\mathrm{Hom}_{\mathbf{Rel}}(A, B) = 2^{A\times B}$, i.e. the morphisms from A to B are all binary relations between A and B. Note that this includes all total and even partial functions from A to B. It also means that **Rel** does not have elements for simple cardinality reasons: $|\mathrm{Hom}_{\mathbf{Rel}}(E, T)| = 2^{|E|\cdot|T|}$. The composition of $A \xrightarrow{f} B$ and $B \xrightarrow{g} C$ is $g \circ f := \{(a, c) \in A \times C | \exists b \in B : (a, b) \in f \text{ and } (b, c) \in g\}$, which is indeed associative. The identity morphism Id_A is the identity function $\{(a, a)|a \in A\}$.

We want to adapt the general pattern laid out in Section 4.6 to our new category **Rel**. Suppose we are given sets T, Y, relation $f : T \times T \to Y$ and relation $\alpha : Y \to Y$. We will let two objects T, Y in **Rel** take the role of the respective sets. When converting f into the categorical realm we are confronted with a problem. **Rel** has products but they are not at all set-theoretic products. In fact, $A \times B$ in **Rel** is the disjoint, set-theoretic union $A \uplus B$. It is not clear how, in general, a relation f with the set-theoretic product $T \otimes T$ as its domain can be faithfully converted to one with domain $T \uplus T$. We therefore use a morphism $T \otimes T \xrightarrow{f} Y$ from the cartesian product $T \otimes T$ to Y and a morphism $Y \xrightarrow{\alpha} Y$. We also define the morphism $T \xrightarrow{\Delta} T \otimes T$ to be the diagonal function: $\Delta := \{(t, (t, t))|t \in T\}$.

Now to choose an object E and a function $e : T \to \mathrm{Hom}_{\mathbf{Rel}}(E, T)$ that embeds elements of T into the set of relations between E and T. In general, $|\mathrm{Hom}_{\mathbf{Rel}}(A, B)| = \left|2^{A\times B}\right| = 2^{|A|\cdot|B|}$ so $E := \{*\}$ already suffices in terms of size: $\mathrm{Hom}_{\mathbf{Rel}}(E, T)$ has size $2^{|T|}$ and $|T|$ would be enough. We then let $e(t) := \{(*, t)\}$.

In formulating an analogon to Eq. (3) we are again confronted with the problem of the categorical product. We must find a substitute $E\otimes T \xrightarrow{k} T\otimes T$ for $E\times T \xrightarrow{(e(t)\times \mathrm{Id}_T)} T\times T$ as well as a substitute $T \xrightarrow{\imath'} E\otimes T$ for $T \xrightarrow{\imath} E\times T$. We make the obvious choices of $k_t := \{((*, s), (t, s))|s \in T\}$ and $\imath' := \{(s, (*, s))|s \in T\}$. We then see that for any $t \in T$, from

$$\alpha \circ f \circ \Delta = f \circ k_t \circ \imath' \tag{14}$$

would follow

$$\alpha \circ f \circ \Delta \circ e(t) = f \circ k_t \circ \imath' \circ e(t). \tag{15}$$

Because we clearly have

$$k_t \circ \imath' \circ e(t) = \Delta \circ e(t) = \{(*, (t, t))\} \tag{16}$$

we can then conclude

$$\alpha \circ f \circ \Delta \circ e(t) = f \circ \Delta \circ e(t). \tag{17}$$

From this we hope to conclude something about the interaction between t, f and α. We cannot talk about fixpoints in this context yet, since that concept is not defined for relations.

Let us apply this to the recursion theorem. Suppose we are given a total function $F : \mathbb{N} \to \mathbb{N}$. Of course $T = \mathbb{N}$, $Y = \mathscr{C}^1$ and f is the partial function (and therefore relation) $\{(m, \phi_{\phi_m(n)})|n \in \mathrm{dom}\, \phi_m\}$. In contrast with **Set**$*$ our current category **Rel** *can* define α explicitly: $\alpha := \{(\phi_x, \phi_{F(x)})|\phi_x \in \mathscr{C}^1\}$. When $x \neq x'$ collide then $\phi_x = \phi_{x'}$ simply relates to both $\phi_{F(x)}$ and $\phi_{F(x')}$ under α.

It turns out that Eq. (14) does not in general hold for any $t \in T$. Note that $f \circ \Delta = \{(m, \phi_{\phi_m(m)}) | m \in \operatorname{dom} \phi_m\}$. Since the enumeration ϕ has many collisions $\phi_{\phi_m(m)} = \phi_{\phi_{m'}(m')}$ we see that $\alpha \circ f \circ \Delta = \{(m, \phi_{F(\phi_{m'}(m'))}) | m \in \operatorname{dom} \phi_m \wedge m' \in \operatorname{dom} \phi_{m'} \wedge \phi_{\phi_m(m)} = \phi_{\phi_{m'}(m')}\}$. Meanwhile $f \circ k_t \circ \imath' = \{(m, \phi_{\phi_t(m)}) | m \in \operatorname{dom} \phi_t\}$. For these last two relations to be equal we therefore need for all $m \in \mathbb{N}$ that

- $m \in \operatorname{dom} \phi_t$ iff $m \in \operatorname{dom} \phi_m$ and
- if $m \in \operatorname{dom} \phi_t$ and $m \in \operatorname{dom} \phi_m$ then for all $m' \in \mathbb{N}$ with $\phi_{\phi_{m'}(m')} = \phi_{\phi_m(m)}$ we have $\phi_{F(\phi_{m'}(m'))} = \phi_{\phi_t(m)}$.

This provides us with enough information to essentially reproduce the counterexample from Example 38, this time to prove there can be no t for which Eq. (14) holds.

Example 41. *Let $F(n) := n + 1$ and suppose that Eq. (14) holds for some t.*

We will prove by induction that when $\phi_n = \phi_{n'}$ then $\phi_{n+k} = \phi_{n'+k}$ for any $k \in \mathbb{N}$.

Let $n_1 < n_2$ be a colliding pair, i.e. $\phi_{n_1} = \phi_{n_2}$. Such n_1, n_2 exist since every partial computable function has infinitely many indices in the enumeration $(\phi_n)_{n \in \mathbb{N}}$. Assuming the induction proof we can conclude that starting from n_1, ϕ repeats the same functions over and over again in a cycle of length at most $n_2 - n_1$. The surjectivity of the enumeration contradicts the fact that there are infinitely many partial computable functions.

Now for the induction proof. The base case $k = 0$ is trivial and we will prove that if $\phi_n = \phi_{n'}$ then $\phi_{n+1} = \phi_{n'+1}$. Suppose that $\phi_n = \phi_{n'}$. Then there exist some m and m' such that $\phi_m(m) = n$ and $\phi_{m'}(m') = n'$. Since $\phi_{\phi_m(m)} = \phi_{\phi_{m'}(m')}$, by Eq. (14) we have that $\phi_{F(\phi_m(m))} = \phi_{\phi_t}(m) = \phi_{F(\phi_{m'}(m'))}$. However $\phi_{F(\phi_m(m))} = \phi_{\phi_m(m)+1} = \phi_{n+1}$ and $\phi_{F(\phi_{m'}(m'))} = \phi_{\phi_{m'}(m')+1} = \phi_{n'+1}$ so $\phi_{n+1} = \phi_{n'+1}$.

So far we have seen that, similarly to in Section 4.6, the first equation, the analogon to Eq. (3), is not true. Worse than in **Set**$*$, however, with our **Rel** definitions Eq. (17), the analogon to Eq. (6), is not even true, unless in trivial cases. Note that the left-hand side $\alpha \circ f \circ \Delta \circ e(t)$ of Eq. (17) is equal to the singleton or empty set $\{(*, \phi_{F(\phi_m(m))}) | t \in \operatorname{dom} \phi_t \wedge m \in \operatorname{dom} \phi_m \wedge \phi_{\phi_m(m)} = \phi_{\phi_t(t)}\}$ and the right-hand side $f \circ \Delta \circ e(t)$ is equal to the singleton or empty set $\{(*, \phi_{\phi_t}(t)) | t \in \operatorname{dom} \phi_t\}$. If ϕ_t is not defined on t then Eq. (17) is obviously and trivially true because both sides are the empty set. If ϕ_t *is* defined on t, however, then Eq. (17) implies that for all $m \in \operatorname{dom} \phi_m$ with $\phi_{\phi_m(m)} = \phi_{\phi_t(t)}$ we have $\phi_{F(\phi_m(m))} = \phi_{\phi_t}(t)$. We can again prove that this does not hold for the same F as Example 38 and Example 41, though using a different technique.

Example 42. *Let $F(n) := n + 1$ and suppose that Eq. (17) holds for some t. Suppose that $t \in \operatorname{dom} \phi_t$.*

We prove by induction that $\phi_{F^k(\phi_t(t))} = \phi_{\phi_t(t)}$ for all k, where F^k stands for k-fold application of F. The base case $k = 0$ is trivially true. Suppose $\phi_{F^k(\phi_t(t))} = \phi_{\phi_t(t)}$. We show that $\phi_{F^{k+1}(\phi_t(t))} = \phi_{\phi_t(t)}$. We first realize that there is an m such that $\phi_m(m) = F^k(\phi_t(t))$. Therefore $\phi_{\phi_m(m)} = \phi_{F^k(\phi_t(t))} = \phi_{\phi_t(t)}$ by the induction hypothesis. This allows us to rewrite the $k+1$-fold application: $\phi_{F^{k+1}(\phi_t(t))} = \phi_{F(F^k(\phi_t(t)))} = \phi_{F(\phi_m(m))} = \phi_{\phi_t(t)}$, where the last equality uses Eq. (17).

This means that starting from $\phi_{\phi_t(t)}$, the enumeration ϕ repeats the same single function ad infinitum, rendering the range of ϕ finite. The surjectivity of the enumeration contradicts the fact that there are infinitely many partial computable functions.

We conclude that this approach in the category **Rel** of relations does not readily allow us to prove the recursion theorem. **Rel**, like **Set**$*$ does not have elements. Worse, though, its categorical product makes it difficult to define f. We circumvented this by forgoing the categorical product altogether and using the set-theoretic product. This also altered the signatures of $E \times T \overset{(e(t) \times \mathrm{Id}_T)}{\rightarrow} T \times T$ and $T \overset{\imath}{\rightarrow} E \times T$. Using these, we found analoga to Eq. (3) through Eq. (6) (namely Eq. (14) through Eq. (17), respectively). Like was the case with **Set**$*$ in Section 4.6, we cannot prove Eq. (14). This time, though, we are even able to properly refute it: there is an F for which Eq. (14) does not hold for any t. It is unclear how Eq. (17) (for some t) would help us find a fixpoint of α. This is, however, quite irrelevant as, in contrast with Section 4.6, Eq. (17) can also be refuted.

Appendices

A WHILE program for the Ackermann function

The following listing is a Python 3 program which, when given two integer numbers m and n, outputs $\psi(m, n)$. Besides some bookkeeping to check for invalid input and to interact with the operating system, it makes use only of the following programming concepts in the main loop. It is clear that these concepts can be implemented by, e.g., a URM program.

- Variables global to the whole loop. Their number is known in advance so they can be stored in registers.
- Simple arithmetic.
- `if-then-else` clauses depending on natural number comparison. This can be implemented with the J instruction.
- One `while` loop. This can be implemented with the J instruction.
- One stack. The Python program uses a list for implementation. This provides a convenient interface, including the ability to push and pop, to peek and alter elements a given number of steps away from the top, and to measure the size of the stack. On the URM, the stack is not to be implemented by having every element in a separate register, since any URM program can only reference a pre-determined number of registers and the stack can grow arbitrarily. Instead, we make use of the unique factorization theorem, which uniquely encodes a finite string of natural numbers into one natural number, and store the entire stack in one register.

We might very well need many additional registers, but only a fixed number of them.

```
#!/bin/python

import sys

stack = []

def main():

    # when stack contains one element, that is the final value
    while len(stack) > 1:

        # internalize our stack frame
        returnvalue = stack[-1]
        pc = stack[-2]
        n = stack[-3]
        m = stack[-4]

        # increase our program counter
        stack[-2] += 1

        # determine which case of the function by parts we are in
        if m == 0:
            # case 1
            # remove our stack frame
            stack.pop()
            stack.pop()
            stack.pop()
            stack.pop()
            # put the correct value in the returnvalue below
            stack[-1] = n+1

        elif n == 0:
            # case 2

            if pc == 0:
                # we must run psi(m-1, 1)
                stack.append(m-1)
                stack.append(1)
                stack.append(0)
                stack.append(None)

            elif pc == 1:
                # we just ran psi(m-1, 1) and must report the returnvalue
                # remove our stack frame
                stack.pop()
                stack.pop()
                stack.pop()
                stack.pop()
                # put the correct value in the returnvalue below
                stack[-1] = returnvalue

            else:
                print("error: pc == 2 in case 2")
                return

        else:
            # case 3

            if pc == 0:
                # we must run psi(m, n-1)
                stack.append(m)
                stack.append(n-1)
                stack.append(0)
                stack.append(None)

            elif pc == 1:
                # we just ran psi(m, n-1) and must use it to run psi(m-1, psi(m, n-
    1))
                stack.append(m-1)
                stack.append(returnvalue)
```

```
                stack.append(0)
                stack.append(None)

            elif pc == 2:
                # we just ran psi(m-1, psi(m, n-1)) and must report the returnvalue
                # remove our stack frame
                stack.pop()
                stack.pop()
                stack.pop()
                stack.pop()
                # put the correct value in the returnvalue below
                stack[-1] = returnvalue

            else:
                print("error: pc == 2 in case 3")
                return

    final = stack.pop()
    print(final)

if __name__ == "__main__":
    m = None
    n = None
    try:
        m = int(sys.argv[1])
        n = int(sys.argv[2])
    except:
        print("Could not read two integers")
        quit()

    if m < 0 or n < 0:
        print("Arguments should be at least 0")
        quit()

    # the final return value
    stack.append(None)

    # the first stack frame
    stack.append(m)
    stack.append(n)
    stack.append(0)
    stack.append(None)

    #run the loop
    main()
```

References

[Ack28] Wilhelm Ackermann. “Zum Hilbertschen Aufbau der reellen Zahlen”. In: *Math. Ann.* 99.1 (Dec. 1928), pp. 118–133. ISSN: 0025-5831, 1432-1807. DOI: 10.1007/BF01459088. URL: http://link.springer.com/10.1007/BF01459088 (visited on 11/09/2023).

[BM72] Paul Bratley and Jean Millo. “Computer recreations”. In: *Software: Practice and Experience* 2.4 (1972), pp. 397–400. DOI: https://doi.org/10.1002/spe.4380020411. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.4380020411.

[Chu36] Alonzo Church. “An Unsolvable Problem of Elementary Number Theory”. In: *American Journal of Mathematics* 58.2 (Apr. 1936), p. 345. ISSN: 00029327. DOI: 10.2307/2371045. URL: https://www.jstor.org/stable/2371045?origin=crossref (visited on 11/11/2023).

[Cut80] Nigel Cutland. *Computability: An Introduction to Recursive Function Theory.* Cambridge University Press, 1980. DOI: 10.1017/CBO9781139171496.

[Fri58] Richard M. Friedberg. “Three theorems on recursive enumeration. I. Decomposition. II. Maximal set. III. Enumeration without duplication”. In: *Journal of Symbolic Logic* 23.3 (Sept. 1958), pp. 309–316. ISSN: 0022-4812, 1943-5886. DOI: 10.2307/2964290. URL: https://www.cambridge.org/core/product/identifier/S0022481200057996/type/journal_article (visited on 16/10/2023).

[Geo91] Georg Cantor. “Ueber eine elementare Frage der Mannigfaltigkeitslehre”. In: *Jahresbericht der Deutschen Mathematiker-Vereinigung* (1891), pp. 75–78. URL: https://www.digizeitschriften.de/dms/img/?PID=GDZPPN002113910 (visited on 07/09/2023).

[Kle38] S. C. Kleene. “On Notation for Ordinal Numbers”. In: *The Journal of Symbolic Logic* 3.4 (1938). Publisher: Association for Symbolic Logic, pp. 150–155. ISSN: 00224812. URL: http://www.jstor.org/stable/2267778 (visited on 19/11/2023).

[Kum90] Martin Kummer. “An easy priority-free proof of a theorem of Friedberg”. In: *Theoretical Computer Science* 74.2 (1990), pp. 249–251. ISSN: 03043975. DOI: 10.1016/0304-3975(90)90141-4. URL: https://linkinghub.elsevier.com/retrieve/pii/0304397590901414 (visited on 16/10/2023).

[Law69] F. William Lawvere. “Diagonal arguments and cartesian closed categories”. In: Barry Mitchell et al. *Category Theory, Homology Theory and their Applications II.* Vol. 92. Series Title: Lecture Notes in Mathematics. Berlin, Heidelberg: Springer Berlin Heidelberg, 1969, pp. 134–145. ISBN: 978-3-540-04611-0 978-3-540-36101-5. DOI: 10.1007/BFb0080769. URL: http://link.springer.com/10.1007/BFb0080769 (visited on 17/05/2023).

[Men10] Mendelson, Elliott. *Introduction to Mathematical Logic.* 5th ed. CRC Press, 2010. ISBN: 978-1-58488-876-5. (Visited on 25/10/2023).

[Pét35] Rózsa Péter. “Konstruktion nichtrekursiver Funktionen”. In: *Math. Ann.* 111.1 (Dec. 1935), pp. 42–60. ISSN: 0025-5831, 1432-1807. DOI: 10.1007/BF01472200. URL: https://link.springer.com/10.1007/BF01472200 (visited on 11/09/2023).

[Rob48] Raphael M. Robinson. “Recursion and double recursion”. In: *Bull. Amer. Math. Soc.* 54.10 (1948), pp. 987–993. ISSN: 0273-0979, 1088-9485. DOI: 10.1090/S0002-9904-1948-09121-2. URL: https://www.ams.org/bull/1948-54-10/S0002-9904-1948-09121-2/ (visited on 11/09/2023).

[Rog67] Rogers, Hartley. *Theory of Recursive Functions and Effective Computability.* MIT Press, 1967. 504 pp. ISBN: 978-0-262-68052-3. (Visited on 19/11/2023).

[Tur37] A. M. Turing. “On Computable Numbers, with an Application to the Entscheidungsproblem”. In: *Proceedings of the London Mathematical Society* s2-42.1 (1937), pp. 230–265. ISSN: 00246115. DOI: `10.1112/plms/s2-42.1.230`. URL: `http://doi.wiley.com/10.1112/plms/s2-42.1.230` (visited on 04/11/2023).

[Yan03] Noson S. Yanofsky. “A Universal Approach to Self-Referential Paradoxes, Incompleteness and Fixed Points”. In: *Bulletin of Symbolic Logic* 9.3 (Sept. 2003), pp. 362–386. ISSN: 1079-8986, 1943-5894. DOI: `10.2178/bsl/1058448677`. URL: `https://www.cambridge.org/core/product/identifier/S107989860000442X/type/journal_article` (visited on 12/05/2023).

[Yan22] Noson S. Yanofsky. *Theoretical Computer Science for the Working Category Theorist.* 1st ed. Cambridge University press, 31st Mar. 2022. ISBN: 978-1-108-87234-8 978-1-108-79274-5. DOI: `10.1017/9781108872348`. URL: `https://www.cambridge.org/core/product/identifier/9781108872348/type/element` (visited on 12/05/2023).