A Constructive and Synthetic Theory of Reducibility: Myhill's Isomorphism Theorem and Post's Problem for Many-one and Truth-table Reducibility in Coq (Full Version)

Yannick Forster, Felix Jahn, Gert Smolka

A Constructive and Synthetic Theory of Reducibility

Myhill's Isomorphism Theorem and Post's Problem for Many-one and Truth-table Reducibility in Coq (Full Version)

Yannick Forster
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
Inria, Gallinette Project-Team, Nantes, France

Felix Jahn
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

Gert Smolka
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

We present a constructive analysis and machine-checked synthetic approach to the theory of one-one, many-one, and truth-table reductions carried out in the Calculus of Inductive Constructions, the type theory underlying the proof assistant Coq.

In synthetic computability, one assumes axioms allowing to carry out computability theory with all definitions and proofs purely in terms of functions of the type theory with no mention of a model of computation. Our synthetic proof of Myhill's isomorphism theorem that one-one equivalence yields a computational isomorphism makes a compelling case for synthetic computability due to its simplicity without sacrificing formality.

Synthetic computability also clears the lense for constructivisation. We do not assume classical axioms, not even Markov's principle, possible by a careful constructivised definition of simple and hypersimple predicates, still yielding the expected strong results: a simple predicate exists, is enumerable, undecidable, but many-one incomplete (Post's problem for many-one reducibility), and a hypersimple predicate exists, is enumerable, undecidable, but truth-table incomplete (Post's problem for truth-table reducibility).

2012 ACM Subject Classification Theory of computation → Constructive mathematics; Type theory

Keywords and phrases type theory, computability theory, constructive reverse mathematics, Coq

Supplementary Material `github.com/uds-psl/coq-synthetic-computability/tree/degrees`.

Funding *Yannick Forster*: received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101024493.

1 Introduction

The founding moment of "computability theory" deserving of the suffix "theory" was maybe Emil L. Post's 1944 paper [18]. Post introduced the concepts of one-one, many-one, and truth-table reducibility and identified and answered important questions on the structure of the reducibility degrees induced by these relations. Centrally, Post was interested in the question whether there are enumerable but undecidable degrees such that an undecidability proof cannot be done by reduction from the halting problem. For many-one and truth-table reducibility, Post was able to construct such degrees by introducing simple and hypersimple sets, which are still taught in modern textbook presentations of the field. The question whether an enumerable, undecidable problem which is not Turing-reducible from the halting problem exists became known as *Post's problem*, and we reuse the terminology *Post's problem for many-one reducibility* ($\preceq_m$) and *Post's problem for truth-table reducibility* ($\preceq_{\mathrm{tt}}$) in the same fashion. Early in his paper, Post remarks *'That mathematicians generally are oblivious to the importance of this work of Gödel, Church, Turing, Kleene, Rosser and others as it affects the subject of their own interest is in part due to the forbidding, diverse and alien*

formalisms in which this work is embodied.'

The evolution of "computability" to "computability theory" started by Post was enabled by a presentation of the lead questions in a more appealing, intuitive way, abstracting away from the "forbidding, alien formalisms" constituted by general recursive functions, Turing machines, the λ-calculus, or Post's own tag systems. However, concrete formalisations of results are still omnipresent in Post's work and are just hidden from the reader. Post remarks that for his paper *'in every instance the informal "proof" was first obtained; and once gotten, transforming it into the formal proof turned out to be a routine chore'*. In computability theory, a fully formal proof of course elaborates on all logical details as in other areas of mathematics, but it also has to explicitly construct (Gödel codes of) programs in the chosen model of computation, insert universal machines, apply the *s-m-n* theorem, use fixpoint theorems, etc. Up to the present day the broad field of computability theory, including complexity theory, has not changed in this matter: proofs might be developed formally with pen and paper, but they are only communicated informally.

We work in a different setting: synthetic computability using as logical foundation the Calculus of Inductive Constructions (CIC, the type theory underlying the Coq proof assistant) following Forster [7]. In the most basic form of synthetic computability, morally present in the Russian school of constructivism due to Markov [13], one assumes an axiom stating that for every function $\mathbb{N} \to \mathbb{N}$ there exists a μ-recursive function computing it, known as CT (Church's thesis, [11]). In synthetic computability due to Richman [20], a function $\phi\colon \mathbb{N} \to (\mathbb{N} \rightharpoonup \mathbb{N})$ is *abstractly* assumed to be universal for all functions $\mathbb{N} \rightharpoonup \mathbb{N}$ (the axiom EPF, stating that $\forall f : \mathbb{N} \rightharpoonup \mathbb{N}.\exists c.\ \phi_c \equiv f$). Richman assumes the axiom of countable choice to be able to prove the *s-m-n* theorem, and in his book with Bridges [2] they remark that assuming just an *s-m-n* operator abstractly would also suffice, without elaborating further. Abstracting even more, Bauer [1] just works with enumerable sets and assumes that the set of enumerable sets is enumerable, together with Markov's principle and the axiom of countable choice.

As a consequence of assuming axioms of synthetic computability together with the axioms of countable choice, the law of excluded middle becomes disprovable. This renders synthetic computability in these settings a constructivist, anti-classical endeavour. Textbook presentations of computability theory are however explicit in that they use classical logic freely. Thus, to translate textbooks presentations into an anti-classical synthetic setting, one has to constructivise proofs on the go. However, this is not always completely possible: It is well-known that Post's theorem that a set is decidable if and only if both it and its complement are enumerable is known to be equivalent to MP [26]. For other results like Myhill's isomorphism theorem, hypersimple sets, and Post's problem to the best of our knowledge it was unknown which logical assumptions such as LEM or MP are necessary.

Forster [7, 6] observes that in CIC, assuming a universal function ϕ abstractly together with an abstract *s-m-n* operator suffices to carry out synthetic computability theory. Equivalently, several axioms are presented: The axiom EPF, concerned with partial functions and more akin to the one used by Richman, and the axiom EA, concerned with enumerable predicates and more akin to Bauer's axiom. Since in CIC the axiom of countable choice is not provable, it seems that one can consistently assume classical logical axioms even as strong as the law of excluded middle if needed.

The setting is fully in the spirit of Post's abstraction away from the "forbidding, alien formalisms" of machine models and forms a suitable setting for the analysis of the constructive status of theorem in computability theory. The present paper

1. presents a fully synthetic development of Myhill's isomorphism theorem [14] and most

results from Post's 1944 paper [18], with no reference to any model of computation,

2. uses intuitionistic logic enriched with the assumption of a universal enumerator and the *s-m-n* theorem and *no* additional axioms, and
3. is fully mechanised in the Coq proof assistant as a further step in the overarching goal of mechanising the pillar-stones of mathematics and computer science. The results of the paper are hyperlinked with Coq code.

Since formalisation is a necessary pre-requisite for mechanisation, mechanisations of computability theory based on a textboook approach do not carry far: Virtually all models of computation are hard to formalise and even harder to mechanise, and although degrees of tedium vary, no model is as pleasing for mechanisation as the synthetic approach.

For the present paper, converting the well-known proofs from a set-theoretic foundation present in textbooks to type theory was not problematic. No extensionality issues were encountered, and predicates work well for the formalisation of decision problems.

Most prominently, we give machine-checked synthetic proofs of:

1. Myhill's isomorphism theorem: One-one equivalent predicates are isomorphic.
2. Post's problem for $\preceq_m$: an enumerable, undecidable, many-one incomplete, simple predicate exists.
3. Post's problem for $\preceq_{\mathsf{tt}}$: an enumerable, undecidable, truth-table incomplete, hypersimple predicate exists.

Myhill's isomorphism theorem is a beautiful and concise case study of synthetic computability, needing no axioms whatsoever. We believe that our version is superior to both formal and informal textbook presentations, while being fully mechanised in less than 300 LoC.

For Post's problem for $\preceq_m$, the definition of a simple predicate in the synthetic setting is most interesting. The complement of a simple predicate has to be infinite, but is not allowed to have an infinite, enumerable subpredicate. In classical mathematics, p is infinite if and only if it is Cantor-infinite, i.e. if there is an injective function $\mathbb{N} \hookrightarrow p$. However, Cantor-infinite predicates (defined via functions) have a (synthetically) enumerable, infinite subpredicate – enumerated by the function witnessing Cantor infinity. In constructive mathematics, a predicate p is called infinite if for any sequence $[x_1, \dots, x_n]$ there exists a y different from all x_i but s.t. py. However, any fully constructive proof of infinity in this sense can be turned into a proof of Cantor infinity, meaning there can be no such proof for the complement of a simple predicate. It is thus crucial that infinity is defined to be exactly non-finiteness. The complement of a simple predicate is then non-finite, but not (synthetically) Cantor-infinite. Only with this definition of infinity Post's problem for $\preceq_m$ can be settled constructively.

For Post's problem for $\preceq_{\mathsf{tt}}$ several interesting aspects appear: First, the definition of hypersimple predicates has to be chosen carefully to ensure that hypersimple predicates are tt-incomplete. The construction of a hypersimple predicate H is then easier. But since conventional proofs of its undecidability factor via simpleness (and since the textbook proofs showing that hypersimple predicates are simple seem to be inherently classical and we only manage to weaken the assumption to MP), we give a direct, fully constructive undecidability proof for H, which however does not generalise to arbitrary hypersimple predicates.

We try to give intuitive conceptual outlines in the paper, but focus on the interesting synthetic and constructive aspects more than the proof ideas. We largely follow the excellent book by Rogers [21], supplemented by the books by Cutland [3], Soare [22], and Odifreddi [15]. The Bachelor's thesis of the second author [10], containing preliminary results covered in this paper, discusses some aspects in more detail. In general, it might be helpful for non-experts in computability theory to consult one of the books in cases where the presented intuition is not sufficient.

2 The Calculus of Inductive Constructions

We work in the polymorphic calculus of cumulative inductive constructions as implemented by the Coq proof assistant [24], which we will refer to as "CIC". The calculus is a constructive type theory with a (cumulative hierarchy of) type universe(s) $\mathbb{T}$ and an impredicative universe of propositions $\mathbb{P} \subseteq \mathbb{T}$. The inductive types of interest in this paper are:

$$\begin{aligned}
n : \mathbb{N} &::= 0 \mid \mathsf{S}\, n && \text{(natural numbers)}\\
b : \mathbb{B} &::= \mathsf{false} \mid \mathsf{true} && \text{(booleans)}\\
o : \mathbb{O}A &::= \mathsf{None} \mid \mathsf{Some}\, a \quad \textit{where } a : A && \text{(options)}\\
l : \mathbb{L}A &::= [] \mid a :: l \quad \textit{where } a : A && \text{(lists)}\\
A + B &:= \mathsf{inl}\, a \mid \mathsf{inr}\, b \quad \textit{where } a : A \textit{ and } b : B && \text{(sums)}\\
A \times B &:= (a, b) \quad \textit{where } a : A \textit{ and } b : B && \text{(pairs)}\\
\Sigma x : X. Ax &:= (x_0, a) \quad \textit{where } A\colon X \to \mathbb{T},\ x_0 : X, a : Ax_0 && \text{(dependent pairs)}
\end{aligned}$$

One can construct a pairing function $\langle \_ , \_ \rangle : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ and for all $f : \mathbb{N} \to \mathbb{N} \to X$ an inverse construction $\lambda\langle n, m\rangle.\, fnm$ of type $\mathbb{N} \to X$ s.t. $(\lambda\langle n, m\rangle.fnm)\langle n, m\rangle = fnm$.

We write $n =_{\mathbb{B}} m$ for boolean equality on $\mathbb{N}$ and $\neg_{\mathbb{B}}$ for boolean negation. We write **if** o **is** $\mathsf{Some}\, x$ **then** ... **else** ... for a case analysis on options, and similarly for other types.

We use the standard functions $\mathsf{map}\colon (X \to Y) \to \mathbb{L}X \to \mathbb{L}Y$, $\_ +\!\!+ \_ : \mathbb{L}X \to \mathbb{L}X \to \mathbb{L}X$, $\mathsf{filter}\colon (X \to \mathbb{B}) \to \mathbb{L}X \to \mathbb{L}X$, and $\_[\_] : \mathbb{L}X \to \mathbb{N} \to \mathbb{O}X$. We may write $|l|$ for the length of $l : \mathbb{L}X$ and $[fx \mid x \in l, gx]$ for $\mathsf{map}\ f\ (\mathsf{filter}\ g\ l)$.

For $p\colon X \to \mathbb{N}$, we denote the complement as $\overline{p}$ and the Cartesian product with $q\colon X \to \mathbb{N}$ as $p \times q$. p is called *inhabited* if $\exists x.px$. We lift those notions to types by identifying a type X with $\lambda(x : X).\top$. We write $\mathsf{Forall}_2\, p\, l_1\, l_2$ where $l_1 : \mathbb{L}X$, $l_2 : \mathbb{L}Y$ and $p\colon X \to Y \to \mathbb{P}$ if p holds pointwisely on the lists l_1 and l_2.

Many results rely on the following elimination principle, reminiscent of the μ operator of general recursive functions.

Fact 1. *There is a guarded minimisation function*

$$\begin{aligned}
&\mu_{\mathbb{N}} : \forall f : \mathbb{N} \to \mathbb{B}.\ (\exists n.\ fn = \mathsf{true}) \to\\
&\qquad \Sigma n.\ fn = \mathsf{true} \wedge \forall m.\ fm = \mathsf{true} \to m \geq n.
\end{aligned}$$

We speak $\Sigma n.pn$ as "one can construct n s.t. pn". An overview on provable and non-provable elimination principles can be found in Section 2.

The universe of propositions $\mathbb{P}$ is impredicative, so e.g. $(\forall X : \mathbb{P}.\ X) : \mathbb{P}$, and a sub-universe of $\mathbb{T}$, i.e. whenever $P : \mathbb{P}$ we also have $P : \mathbb{T}$. Both aspects however only play a minor role in this paper. The important aspect is that the universe $\mathbb{P}$ is separated from the universe $\mathbb{T}$. That means that in general, computations cannot do case analysis on proofs and return a computational value.

Most instructively, the difference can be explained by looking at dependent sums $\Sigma x.\ Ax$ and existential quantification $\exists x.\ Ax$. Both are implemented as dependent pairs via an inductive type, with the only difference that $(\exists x.\ Ax) : \mathbb{P}$ but $(\Sigma x.\ Ax) : \mathbb{T}$. Dependent sums can be eliminated in arbitrary contexts, i.e. there is an elimination function of type

$$\forall p : (\Sigma x.\ Ax) \to \mathbb{T}.\ (\forall xy.\ p(x, y)) \to \forall (s : \Sigma x.\ Ax).\ ps.$$

In contrast, existential quantification can only be eliminated for $p : (\exists x.Ax) \to \mathbb{P}$, i.e.

$$\forall p : (\exists x.\ Ax) \to \mathbb{P}.\ (\forall xy.\ p(x, y)) \to \forall (s : \exists x.\ Ax).\ ps.$$

We call such a principle eliminating a proposition into arbitrary types a *large elimination principle*, following the terminology "large elimination" for Coq's case analysis construct `match` [16].

Crucially, CIC proves a large elimination principle for the falsity proposition $\bot$, i.e. explosion applies to arbitrary types: $\forall A : \mathbb{T}.\ \bot \to A$. For existential quantification, a large elimination principle is not provable.

There are two exceptions to this observation: First, as seen in Fact 1 and Corollary 6, for certain p (p has a decider, p has an enumerator), an elimination principle is provable.

Secondly, whenever an existential quantification $\exists x.\ Ax$ (also in the relational form $\forall x_1.\exists x_2.\ Rx_1x_2$), can be proved *without assumptions*, one could instead prove the stronger result that $\Sigma x.\ Ax$ (for the relational form $\forall x_1.\Sigma x_2.\ Rx_1x_2$ or equivalently $\Sigma f.\forall x_1.\ Rx_1(fx_1)$).

3 Constructive proofs

A proposition $P : \mathbb{P}$ is *stable* if it is unchanged under double negation, i.e. $\neg\neg P \to P$. Furthermore, we say that P is *logically decidable*, if $P \vee \neg P$ holds.

Fact 2. *Logically decidable propositions are stable.*

The *law of excluded middle* LEM states that every proposition is logically decidable:

$\mathsf{LEM} := \forall P : \mathbb{P}.\ P \vee \neg P$

LEM is routinely used in textbooks, often in the form of double negation elimination:

Fact 3. $\mathsf{LEM} \leftrightarrow \forall P : \mathbb{P}.\ \neg\neg P \to P$

It is folklore that LEM is independent in CIC: It can be consistently assumed, but not proved. However, when the conclusion is stable, LEM is not needed for case analysis:

Fact 4. *Let Q be stable. The following hold.*

1. $P \to \neg\neg P$
2. $\neg\neg P \to (P \to Q) \to Q$
3. $(P \vee \neg P \to Q) \to Q$

Markov's Principle (MP) is a consequence of LEM accepted for instance in Russian constructivism and states that satisfiability of a boolean test on natural numbers is stable:

$\mathsf{MP} := \forall f : \mathbb{N} \to \mathbb{B}.\ \neg\neg(\exists n.\ fn = \mathsf{true}) \to (\exists n.\ fn = \mathsf{true})$

MP is also independent in CIC [17, 12]. For classical theorems on enumerable predicates, often full LEM is not needed and MP suffices, due to the following:

Fact 5 ([8] 2.17, [5] 41)**.** *The following are equivalent:*

1. MP
2. $\forall p{:}\,\mathbb{N}{\to}\mathbb{P}.\ \mathcal{S}p \to \forall x.\ \neg\neg px \to px$
3. $\forall p{:}\,X{\to}\mathbb{P}.\ \mathcal{E}p \to \neg\neg(\exists n.\ pn) \to \exists n.\ pn$

4 Synthetic Computability Theory

The key ingredients for synthetic computability theory are synthetic definitions of central notions and suitable synthetic axioms. We define synthetic decidability, enumerability, semi-decidability, and many-one reducibility [8, 5], mirroring the textbook definitions without the requirement that the witnessing function is computable in a model of computation. We recall the synthetic setting due to Forster [7] in which we work.

A predicate $p\colon X\to\mathbb{P}$ is

- *decidable* if there exists a decider: $\mathcal{D}p := \exists f\colon X \to \mathbb{B}.\forall x.\ px \leftrightarrow fx = \mathsf{true}$
- *semi-decidable* if there exists a semi-decider: $\mathcal{S}p := \exists f.\forall x.\ px \leftrightarrow \exists n. fxn = \mathsf{true}$
- *enumerable* if there exists an enumerator: $\mathcal{E}p := \exists f\colon \mathbb{N} \to \mathbb{O}X.\forall x.\ px \leftrightarrow \exists n.\ fn = \mathsf{Some}\, x$
- *strongly enumerable* if there exists a strong enumerator: $\mathcal{E}_+p := \exists f\colon \mathbb{N}\to X.\forall x.\ px \leftrightarrow \exists n.\ fn = x$

We treat n-ary predicates as unary via (implicit) uncurrying. A type X is *discrete* if $\mathcal{D}(\lambda xy\colon X.x=y)$ and *enumerable* if $\mathcal{E}(\lambda x : X.\top)$.

We sum up the usual connections between the notions in Appendix A. The following strengthening of $\mu_{\mathbb{N}}$ (Fact 1) will be crucial to the construction of simple predicates:

Corollary 6. *Let $p\colon\mathbb{N}\to X\to\mathbb{P}$ be enumerable and X discrete. Then there exists $\mu_{\mathcal{E}}$: $\forall n.\ (\exists x.\ pnx) \to \Sigma x.\ pnx$.*

In this paper we will define many-one, one-one, and truth-table reducibility $(\preceq_m, \preceq_1, \preceq_{\mathsf{tt}})$. In general, for a reducibility notion $\preceq_r$ we define predicates $p\colon X\to\mathbb{P}$ and $q\colon Y\to\mathbb{P}$ to be *r-equivalent* if $p \equiv_r q := p \preceq_r q \wedge q \preceq_r p$. Informally, we might also refer to the class of predicates q s.t. $p \equiv_r q$ as the r-degree of p, but will not make degrees formal to avoid extensionality assumptions. A predicate $p\colon\mathbb{N}\to\mathbb{P}$ is called *r-complete* if $\mathcal{E}p \wedge \forall q\colon\mathbb{N}\to\mathbb{P}.\ \mathcal{E}q \to q \preceq_r p$.

All notions of reducibility have in common that they form pre-orders (i.e. are reflexive and transitive) and that they transport decidability backwards, i.e. $p \preceq_r q \to \mathcal{D}q \to \mathcal{D}p$.

We now start with the first notion of synthetic reducibility: A function $f : X \to Y$ is a *many-one reduction* from $p\colon X\to\mathbb{P}$ to $q\colon Y\to\mathbb{P}$ if it expresses p in terms of q:

$$p \preceq_m q := \exists f : X \to Y.\forall x.\ px \leftrightarrow q(fx)$$

Fact 7. *1. Many-one reducibility forms a pre-order.*

2. If $p \preceq_m q$ and q is decidable (/semi-decidable/stable) then p is decidable (/semi-decidable/stable).

3. $p \preceq_m q \to \overline{p} \preceq_m \overline{q}$.

Regarding the order structure of $\preceq_m$, we now prove that decidable predicates constitute minima for the class of non-trivial predicates, and that $\preceq_m$ forms an upper semi-lattice:

Fact 8. *Let p be decidable. If $\exists x_1x_2.\ qx_1 \wedge \neg qx_2$, then $p \preceq_m q$.*

Fact 9. *Let $p\colon X\to\mathbb{P}$ and $q\colon Y\to\mathbb{P}$. Then there is a lowest upper bound $p+q\colon X+Y\to\mathbb{P}$ w.r.t. $\preceq_m$: If $p+q(\mathsf{inl}\, x) := px$ and $p+q(\mathsf{inr}\, y) := qy$, then $p+q$ is the join of p and q w.r.t $\preceq_m$, i.e. $p \preceq_m p+q$, $q \preceq_m p+q$, and for all r if $p \preceq_m r$ and $q \preceq_m r$ then $p+q \preceq_m r$.*

In traditional computability theory, a universal machine for the chosen model of computation is central to almost all interesting results. In a synthetic approach to computability where there is no explicit model of computation and the notion of function and computable function are identified, a universal function cannot be defined. Instead, its existence has to be axiomatically assumed.

To develop synthetic computability theory agnostic towards classical axioms like LEM, we assume the enumerability axiom EA [7, 6]. The axiom EA postulates

1. a *universal enumerator* $\varphi\colon \mathbb{N}\to(\mathbb{N}\to\mathbb{O}\mathbb{N})$
2. for all $p\colon \mathbb{N}\to\mathbb{N}\to\mathbb{P}$

$$(\exists f\colon \mathbb{N}\to(\mathbb{N}\to\mathbb{O}\mathbb{N}).\forall i.\ f_i \text{ enumerates } p_i) \to \exists\gamma\colon \mathbb{N}\to\mathbb{N}.\forall i.\ \varphi_{\gamma i} \text{ enumerates } p_i$$

To ease language, we often refer to (2) as EA in this paper.

We list three consequences of EA:

Fact 10. $\forall p\colon \mathbb{N}\to\mathbb{P}.\ \mathcal{E}p \to \exists c.\ \varphi_c \text{ enumerates } p$

Fact 11. *Let X be enumerable and discrete and $p\colon X\times\mathbb{N}\to\mathbb{P}$ be enumerable. Then*

$\exists\gamma\colon X\to\mathbb{N}.\forall x.\ \varphi_{\gamma x}$ *enumerates* $\lambda y.p(x,y)$.

Fact 12. *Let I be enumerable and discrete and $p\colon I\to\mathbb{N}\to\mathbb{P}$. We have*

$$(\exists f\colon I\to(\mathbb{N}\to\mathbb{O}\mathbb{N}).\forall i.\ f_i \text{ enumerates } p_i) \to \exists\gamma\colon I\to\mathbb{N}.\forall i.\ \varphi_{\gamma i} \text{ enumerates } p_i$$

As is common in developments of computability, we start by defining an enumerable, undecidable, m-complete predicate and its diagonal:

$$\mathcal{W}_c x := \exists n.\ \varphi_c n = \mathsf{Some}\, x \qquad \mathcal{K}c := \mathcal{W}_c c$$

We call $\mathcal{W}$ the *universal table* of enumerable predicates, justified by the following property:

Fact 13. $\forall p\colon \mathbb{N}\to\mathbb{P}.\ \mathcal{E}p \leftrightarrow \exists c.\ \forall x.\ \mathcal{W}_c x \leftrightarrow px$

The universal table $\mathcal{W}$ is enumerable, undecidable and m-complete, i.e. takes the role of the halting problem from textbook computability. $\mathcal{K}$ plays a similar role as the self-halting problem, instead of the codes halting on themselves, $\mathcal{K}c$ holds if c is in the range of φ_c.

We start by showing that the complement of $\mathcal{K}$ is not enumerable. Thus $\mathcal{K}$ is undecidable, and undecidability can be transported to $\mathcal{W}$ via a many-one reduction.

Fact 14. $\neg\mathcal{E}\overline{\mathcal{K}}$ *and thus* $\neg\mathcal{D}\overline{\mathcal{K}}$ *and* $\neg\mathcal{D}\mathcal{K}$

Fact 15. $\mathcal{K}\preceq_m\mathcal{W}$

The reduction can be used to transport the negative results from $\mathcal{K}$ to $\mathcal{W}$:

Corollary 16. $\neg\mathcal{E}\overline{\mathcal{W}}$, $\neg\mathcal{D}\overline{\mathcal{W}}$, $\neg\mathcal{D}\mathcal{W}$.

To show the enumerability of both $\mathcal{K}$ and $\mathcal{W}$, we show the enumerability of $\mathcal{W}$ and again transport via the above many-one reduction, this time positively.

Fact 17. $\mathcal{E}\mathcal{W}$ *and thus* $\mathcal{E}\mathcal{K}$.

We now turn towards m-completeness of $\mathcal{W}$, i.e. that all $p\colon X\to\mathbb{P}$ for enumerable, discrete X many-one reduce to $\mathcal{W}$:

Lemma 18. $\mathcal{W}$ *is m-complete.*

Proof. Let p be enumerable by φ_c via Fact 10. Then $\lambda x.(c,x)$ reduces p to $\mathcal{W}$. ◀

Establishing that $\mathcal{K}$ is m-complete as well now for the first time requires the full strength of EA, whereas before the non-parametric Fact 10 would have sufficed.

Lemma 19. $\mathcal{W}\preceq_m\mathcal{K}$

Proof. We obtain the reduction function γ from Fact 12 with $p(x,y)z := \mathcal{W}_x y$. Since $\forall xyz.\ \mathcal{W}_{\gamma(x,y)}z \leftrightarrow \mathcal{W}xy$ we have $\mathcal{W}_x y \leftrightarrow \mathcal{W}_{\gamma(x,y)}(\gamma(x,y)) \leftrightarrow \mathcal{K}(\gamma(x,y))$. ◀

Corollary 20. $\mathcal{W}\equiv_m\mathcal{K}$ *and $\mathcal{K}$ is m-complete.*

5 Finite predicates

We discuss two notions of finiteness: Listable (often Bishop- or $\mathcal{B}$-finite) and exhaustible (sub- or $\widetilde{\mathcal{B}}$-finite) predicates.

A list $l : \mathbb{L}X$ *lists* a predicate $p\colon X\to\mathbb{P}$ if $px \leftrightarrow x \in l$.

$$\mathcal{L}p := \exists l : \mathbb{L}X.\forall x : X.\ px \leftrightarrow x \in l \qquad (\text{“}p \text{ is listable”})$$

Note that l is not allowed to contain elements not fulfilled by the predicate. To ensure finiteness, this is unnecessarily strict, and we can relax the condition.

A list $l : \mathbb{L}X$ *exhausts* a predicate $p\colon X\to\mathbb{P}$ if $px \to x \in l$:

$$\mathcal{X}p := \exists l : \mathbb{L}X.\ \forall x : X.\ px \to x \in l \qquad (\text{“}p \text{ is exhaustible”})$$

Clearly, listability implies exhaustability, but the converse is equivalent to LEM:

Fact 21. *Listable predicates are exhaustible.*

Lemma 22. *Every exhaustible predicate is not not listable.*

Proof. We first prove the lemma that

$$\forall l_0 : \mathbb{L}X.\forall p\colon X\to\mathbb{P}.\neg\neg\exists l'.\forall x.\ x \in l' \leftrightarrow x \in l_0 \wedge px,$$

which follows by induction on l_0.

Now, let l exhaust p and let p be not listable. We have to prove falsity. Using the lemma, we obtain l' s.t. $\forall x.\ x \in l' \leftrightarrow x \in l \wedge px$ and still have to prove falsity. Now l' lists p. Contradiction. ◀

Lemma 23. *If every exhaustible predicate is listable,* LEM *holds.*

Proof. For $P : \mathbb{P}$ we define $p(x : \mathbb{B}) := P$. p is exhaustible because it is exhausted by $[\mathsf{true}, \mathsf{false}]$. If p is listed by l, case analysis on l allows proving $P \vee \neg P$. ◀

Corollary 24. LEM *holds if and only if every exhaustible predicate is listable.*

6 Pigeonhole principles

Pigeonhole principles are omnipresent in discrete mathematics. We will prove three variants of the pigeonhole principle based on duplicate-free lists: Our formulation of the principle is that for any duplicate-free list l_1 longer than a list l_2 one can obtain an element x which is in l_1 but not in l_2 – for different formalisations of “obtain” in CIC[1]:

1. x is computable: $\forall l_1 l_2.\ \cdots \to \Sigma x.\ \ldots$
2. x constructively exists: $\forall l_1 l_2.\ \cdots \to \exists x.\ \ldots$
3. x classically exists: $\forall l_1 l_2.\ \cdots \to \neg\neg\exists x.\ \ldots$

These are exactly the three possible formalisations of “obtain”: A function returning a dependent pair (Σ), a proof of an existential proposition ($\exists$), or of a double-negated existential proposition ($\neg\neg\exists$, equivalently $\neg\neg\Sigma$). Formulating existence as Σ inherently means that the result has to be *computable*, a property unchanged by the assumption of logical axioms like

[1] Admittedly, this formulation is more a “pigeon-less hole principle”, but we use the well-known terminology.

LEM. $\exists$ has to be proved using a (computable) function, but the function cannot be used computationally after the proof. This means that any constructive proof of $\forall x.\exists y.\dots$ not using assumptions could always be turned into a proof of $\forall x.\Sigma y$ and vice versa, but under assumptions and as assumptions the two behave differently, see Section 2 for more details.

We now turn towards proving the principles, which will vary in the requirements on the underlying type X. For the formalisation, we define $\#l : \mathbb{P}$ for a list $l : \mathbb{L}X$ inductively in the expected way to state that l does not contain any duplicates.

Given an equality decider for the base type X it is straightforward to prove the $\forall\Sigma$-version of the pigeonhole principle:

Lemma 25. *Let d decide equality on X and $l_1, l_2 : \mathbb{L}X$. If $\#l_1$ and $|l_1| > |l_2|$, then $\Sigma x.\, x \in l_1 \land x \notin l_2$.*

Proof. By induction on $\#l_1$, with l_2 generalised. The first case is contradictory, since $|[]| = 0 > |l_2|$ is impossible. Let $x \notin l_1$ and $\#l_1$. Using d allows a case analysis whether $x \in l_2$ or $x \notin l_2$: If $x \notin l_2$, the claim is immediate. If $x \in l_2$, the claim follows from the induction hypothesis for $l_2 := \mathsf{filter}(\lambda y.\neg_{\mathbb{B}}(dxy))l_2$ (i.e. for l_2 with x removed). ◀

Note how the $\forall\Sigma$ version depends on computationally removing an element from a list, and thus on an equality decider d. If no such d is available, a removal function is not definable. However, for the $\forall\exists$ and $\forall\neg\neg\exists$ forms of the pigeonhole principle, a removal *function* is not needed. Instead, for the $\forall\exists$ version it suffices to prove that for any list l_0 and any element x_0, there exists ($\exists$) a list with the same elements of l_0, just x_0 removed. This becomes possible provided $x_1 \neq x_2$ is logically decidable for all x_1, x_2.For the $\forall\neg\neg\exists$ version of the pigeonhole principle, consequently a removal principle of the form $\neg\neg\exists l\dots$ suffices, which can be proved fully constructively without assumptions.

To prove the two removal principles,we define a generalised filter predicate $l_0 \supseteq_p l$ w.r.t. a predicate $p\colon X \to \mathbb{P}$ stating that l is exactly the sublist of l_0 with all elements which fulfil p:

$$\frac{}{[] \supseteq_p []} \qquad \frac{px \qquad l_0 \supseteq_p l}{(x :: l_0) \supseteq_p (x :: l)} \qquad \frac{\neg px \qquad l_0 \supseteq_p l}{(x :: l_0) \supseteq_p l}$$

We can prove two existence principles, one assuming that p is logically decidable, and one proving a double negation:

Fact 26. *Let $l_0 : \mathbb{L}X$. Then (1) $\neg\neg\exists l.\, l_0 \supseteq_p l$ and (2) $(\forall x.px \lor \neg px) \to \exists l.\, l_0 \supseteq_p l$.*

The $\forall\exists$ and $\forall\neg\neg\exists$ forms of the pigeonhole principle follow:

Lemma 27. *If $\#l_1$ and $|l_1| > |l_2|$, then $\neg\neg\exists x.x \in l_1 \land x \notin l_2$.*

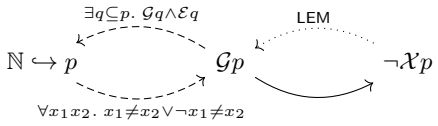
Proof. By induction on $\#l_1$, with l_2 generalised. The case $l_1 = []$ is contradictory since then $|[]| = 0 > |l_2|$. Thus let $x \notin l_1$ and $\#l_1$. Since the claim is negative, we can do a case analysis on $x \in l_2$ by Fact 4 (3). If $x \notin l_2$, the claim is immediate. If $x \in l_2$, we obtain l s.t. $l_2 \supseteq_{(\lambda y.x \neq y)} l$ from Fact 26 (2). The claim follows by induction for l. ◀

Lemma 28. *If $\forall x_1 x_2\colon X.x_1 \neq x_2 \lor \neg x_1 \neq x_2$, $\#l_1$ and $|l_1| > |l_2|$, then $\exists x.\, x \in l_1 \land x\neg \in l_2$.*

Proof. Similar to the last proof, using $x \notin l_2 \lor \neg x \notin l_2$ enabled by Fact 26 (1). ◀

7 Infinite Predicates

The central notions of simple and hypersimple predicates depend on a formalisation of infinity. Similar to finite predicates, there are several classically equivalent but constructively different definitions of infinite predicates. We discuss three possible definitions of infinity, where a predicate $p\colon X\to\mathbb{P}$ is non-finite if it is not finite ($\neg\mathcal{X}p$) (a stable proposition), generative ($\mathcal{G}p$) if for every list there exists a further element fulfilling p not in the list (a $\forall\exists$ proposition), and Cantor-infinite ($\mathbb{N}\hookrightarrow p$) if there exists an injection returning only elements in p (equivalent to a $\forall\Sigma$ proposition). Those three notions of infinite predicates mirror exactly the three different formulations of pigeonhole principles in the previous chapter. We obtain the following graph, where the dashed (dotted) lines are annotated with sufficient (and necessary) conditions:



Formally, a predicate $p\colon X\to\mathbb{P}$ is called *non-finite* if is not finite. A predicate $p\colon X\to\mathbb{P}$ is *finite* (written $\mathcal{L}p$) if $\exists l\colon\mathbb{L}X.\ \forall x.\ px\leftrightarrow x\in l$.

Due to the negation, non-finiteness is equivalent to non-sub-finiteness. A predicate $p\colon X\to\mathbb{P}$ is *subfinite* (written $\mathcal{X}p$) if $\exists l\colon\mathbb{L}X.\ \forall x.\ px\to x\in l$. We discuss more properties on finite predicates in Section 5.

Fact 29. $\neg\mathcal{X}p\leftrightarrow\neg\mathcal{L}p$.

A predicate $p\colon X\to\mathbb{P}$ is called *generative* if for every list there exists a further element fulfilling p which is not in the list:

$$\mathcal{G}p:=\forall l:\mathbb{L}X.\exists x.\ px\wedge x\notin l$$

Fact 30. *Generative predicates are inhabited, and non-finite predicates not not inhabited.*

Generative predicates on $\mathbb{N}$ can be characterised as follows:

Fact 31. $\forall p\colon\mathbb{N}\to\mathbb{P}.\ \mathcal{G}p\leftrightarrow\forall n.\exists m\geq n.\ pm$

The following characterisation of non-finiteness as $\forall\neg\neg\exists$-formula connects the two notions:

Fact 32. *If* $\forall x_1x_2:X.\ x_1=x_2\vee x_1\neq x_2$, *then* $\neg\mathcal{X}p\leftrightarrow\forall l:\mathbb{L}X.\neg\neg\exists x.\ px\wedge x\notin l$.

Corollary 33. $\forall p\colon\mathbb{N}\to\mathbb{P}.\ \neg\mathcal{X}p\leftrightarrow\forall n.\neg\neg\exists m\geq n.pm$

Since the proof of the direction from right to left does not depend on logical decidability of equality we can prove the following:

Fact 34. *Generative predicates are non-finite.*

Fact 35. *If* LEM *holds and p is non-finite, p is generative.*

We introduce a second $\forall\exists$-notion of infinite predicates: $p\colon X\to\mathbb{P}$ is called *unbounded* if there exist duplicate-free lists of arbitrary length containing only elements from p:

$$\mathcal{U}p:=\forall n:\mathbb{N}.\exists l.\ |l|=n\wedge\#l\wedge\forall x\in l.\ px$$

Fact 36. $\neg\mathcal{X}p\leftrightarrow\forall n:\mathbb{N}.\neg\neg\exists l.\ |l|=n\wedge\#l\wedge\forall x\in l.\ px$

Lemma 37. *Generative predicates are unbounded.*

Proof. Let p be generative and $n : \mathbb{N}$. We construct l by induction on n. For $n = 0$ we pick $l = []$. For $n = \mathsf{S}n'$ we use the inductive hypothesis to obtain l, and use generativity to obtain x s.t. px and $x \notin l$. Now $x :: l$ is the wanted list. ◀

We then can prove the following using Lemmas 27 and 28.

Fact 38. *Unbounded predicates are non-finite, and are generative for types X with $\forall x_1 x_2 : X.\ x_1 \neq x_2 \vee \neg x_1 \neq x_2$.*

LEM is necessary for non-finite predicates to be unbounded:

Fact 39. *If all non-finite $p : \mathbb{N} \to \mathbb{P}$ are unbounded,* LEM *holds.*

Corollary 40. *Non-finite predicates are unbounded (/generative) if and only if* LEM *holds.*

Lastly, we introduce Cantor infinity, often used in textbooks. A predicate $p : X \to \mathbb{P}$ is *Cantor-infinite* if there exists an injection $f : \mathbb{N} \to X$ only returning elements in p:

$$\mathbb{N} \hookrightarrow p := \exists f : \mathbb{N} \to X. \forall n_1.\ p(fn_1) \land \forall n_2.\ fn_1 = fn_2 \to n_1 = n_2$$

Fact 41. *Cantor-infinite predicates are unbounded.*

Lemma 42. *Let X be discrete and $p : X \to \mathbb{P}$. Then $\mathbb{N} \hookrightarrow p \leftrightarrow \forall l : \mathbb{L}X. \Sigma x.\ px \land x \notin l$.*

Proof. The forward direction is easy using Lemma 25. Conversely, let $F : \forall l : \mathbb{L}X. \Sigma x.\ px \land x \notin l$ and $fl := \pi_1(Fl)$. Let $g0 := []$ and $g(\mathsf{S}n) := gn +\!\!+ [f(gn)]$. Then pick $\lambda n. f(gn)$. ◀

We have proven that non-finite predicates are generative if and only if LEM holds. However, non-finite, enumerable predicates on $\mathbb{N}$ are generative already given MP:

Lemma 43. *Assume* MP *and let $p : \mathbb{N} \to \mathbb{P}$. If p is enumerable and non-finite, p is generative.*

Proof. Let p be enumerable and non-finite, and let l be given. We have to prove $\exists x.\ x \notin l \land px$. Since p is enumerable, so is $\lambda x.\ x \notin l \land px$. Using Fact 5 it suffices to prove $\neg\neg \exists x.\ x \notin l \land px$, which holds by Fact 32. ◀

We adapt [8, Th. 2.28] to prove that generative, enumerable predicates are Cantor-infinite:

Fact 44. *Let X be discrete. Generative, enumerable predicates over X have a strong, injective enumerator: $\forall p : X \to \mathbb{P}.\ \mathcal{G}p \to \mathcal{E}p \to \exists f.\ f \mathit{injective} \land \forall x.\ px \leftrightarrow \exists n.\ fn = x$.*

Corollary 45. *Generative, enumerable predicates over discrete types are Cantor-infinite.*

In synthetic computability, Cantor-infinite predicates are problematic because the function $f : \mathbb{N} \to X$ can be turned into an enumerator. From $\mathbb{N} \hookrightarrow p$ we cannot conclude that p is enumerable, but that p has a Cantor-infinite, enumerable subpredicate:

Lemma 46. $\mathbb{N} \hookrightarrow p \to \exists q.\ \mathcal{E}q \land (\forall x.\ qx \to px) \land \mathbb{N} \hookrightarrow q$

Proof. Given p and an injection f witnessing $\mathbb{N} \hookrightarrow p$, define $qx := \exists n.\ fn = x$. Clearly, $\forall x.\ qx \to px$ since $\forall n.\ p(fn)$, and $\lambda n.\ \mathsf{Some}\,(fn)$ enumerates p. f still proves $\mathbb{N} \hookrightarrow q$. ◀

Thus, taking Cantor-infinity as notion of infinity is not possible when developing synthetic computability theory. Since any fully constructive proof of generativity could be turned into a proof of Cantor-infinity by Lemma 42, we have that under the assumption of a universal enumerator φ, defining infinity as...

1. Cantor infinity proves there is no simple predicate.
2. generativity makes the existence of simple predicates logically independent.
3. non-finiteness allows to construct a simple predicate.

8 One-one reducibility

We now introduce our second notion of reducibility: One-one reducibility is a special case of many-one reducibility. A function $f\colon X\to Y$ is a *one-one reduction* from p to q if f is an injective many-one reduction from p to q:

$$p \preceq_1 q := \exists f : X \to Y.\ f \text{ is injective} \wedge \forall x.\ px \leftrightarrow q(fx)$$

Fact 47. *1. One-one reducibility forms a pre-order.*
2. If $p \preceq_1 q$ then $p \preceq_m q$.
3. If $p \preceq_1 q$ and q is decidable (/ semi-decidable / stable), then so is p.

It is easy to prove that one-one and many-one reducibility are not equivalent:

Lemma 48. *If $px := \top$ and $qx := x = 0$, $p \preceq_m q$ but $q \not\preceq_1 p$.*

Proof. $\lambda x.0$ is a many-one reduction. Given a one-one reduction f, we need to have $f0 = f1 = 0$. But since f is injective, this would mean that $0 = 1$. Contradiction. ◀

Note that the lemma leaves open whether there are two enumerable, but *undecidable* predicates p and q s.t. $p \preceq_m q$ but $p \not\preceq_1 q$, which is the more interesting case. This question will be settled later via simple predicates.

In Appendix B, we will furthermore show a characterisation of m-reducibility in terms of 1-reducibility via so-called cylindrification, adapting the proof by Rogers [21, §7.6 Th. VIII].

9 Myhill isomorphism theorem

The Myhill isomorphism theorem [14] is a generalisation of the restriction of the Cantor-Bernstein theorem to enumerable, discrete types. In general, the (inherently classical [19]) Cantor-Bernstein constructs a bijection between sets A and B from two injections $A\to B$ and $B\to A$. When restricted to enumerable, discrete types, Cantor-Bernstein becomes fully constructive. As a generalisation, the Myhill isomorphism theorem states that 1-equivalent predicates $p\colon X\to\mathbb{P}$ and $q\colon Y\to\mathbb{P}$ (i.e. there are injective reductions between them, but the reductions are in no relation to each other) are isomorphic (i.e. there are bijective reductions f, g between them s.t. $f(gy) = y$ and $g(fx) = x$).

The isomorphism theorem does not rely on universal machines and can thus be synthetically replicated without axioms.

We prove the Myhill isomorphism theorem (Theorem 52) by loosely following Rogers [21, §7.4 Th. VI], where the isomorphism is constructed in stages. The stages are formed by so-called (finite) *correspondence sequences* between predicates p and q, which are finitary bijections represented as lists $C : \mathbb{L}(X \times Y)$ s.t.

1. $\forall (x, y) \in C.\ px \leftrightarrow qy$
2. $\forall (x, y_1) \in C. \forall (x, y_2) \in C.\ y_1 = y_2$
3. $\forall (x_1, y) \in C. \forall (x_2, y) \in C.\ x_1 = x_2$

We write $x \in_1 C$ if x is an element of the first projection of C, and similarly for $y \in_2 C$.

The crux of the theorem is that for any C as above s.t. $p \preceq_1 q$ and $x_0 \notin_1 C$ one can *compute* y_0 s.t. $(x_0, y_0) :: C$ is a correspondence sequence again, with no condition on p, q:

Lemma 49. *Let f be a one-one reduction from p to q.*

One can construct a function $\mathsf{find}\colon \mathbb{L}(X \times Y)\to X\to Y$ *s.t. if C is a correspondence sequence for p and q and $x_0 \notin_1 C$, then* $\mathsf{find}\ C\ x_0 \notin_2 C$ *and* $px_0 \leftrightarrow q(\mathsf{find}\ C\ x_0)$.

Proof. We first define a function $\gamma : \mathbb{L}(X \times Y) \to X \to X$ recursive in $|C|$:

$$\begin{aligned} \gamma C x &:= x && \text{if } f x \notin_2 C \\ \gamma C x &:= \gamma\,(\mathsf{filter}(\lambda t.t \neq_{\mathbb{B}} (x', fx))\,C)\,x' && \text{if } (x', fx) \in C \end{aligned}$$

For a correspondence sequence C between p and q and $x \notin_1 C$ we have (1) $px \leftrightarrow p(\gamma C x)$, (2) $\gamma C x = x$ or $\gamma X x \in_1 C$, and (3) $f(\gamma C x) \notin_2 C$. The proof is straightforward by induction on the length of C, exploiting the injectivity of f.

Now $\mathsf{find}\, C x_0 := f(\gamma C x_0)$ is the wanted function. ◀

For the rest of this section we fix enumerable, discrete types X and Y s.t. (I_X, R_X) and (I_Y, R_Y) are retractions from X and Y respectively to $\mathbb{N}$ by Fact 95. We construct the isomorphism via a cumulative correspondence sequence:

$$C_0 := [] \qquad C'_n := \begin{cases} (x, \mathsf{find}\, C_n\, x) :: C_n & \text{if } R_X n = \mathsf{Some}\, x \wedge x \notin_1 C_n \\ C_n & \text{otherwise} \end{cases}$$

$$C_{\mathsf{S}n} := \begin{cases} (\mathsf{find}\, \overleftrightarrow{C'_n}\, y, y) :: C'_n & \text{if } R_Y n = \mathsf{Some}\, y \wedge y \notin_2 C_n \\ C'_n & \text{otherwise} \end{cases}$$

where $\overleftrightarrow{C} := \mathsf{map}\,(\lambda(x,y).(y,x))\,C$ is a correspondence sequence for q and p if and only if C is one for p and q.

Fact 50. *C_n is a correspondence sequence for p and q s.t.*

1. $n \leq m \to C_n \subseteq C_m$
2. $I_X x < n \to x \in_1 C_n$
3. $I_Y y < n \to y \in_2 C_n$

C_n now gives rise to the wanted isomorphism:

Lemma 51. *There are functions $f' : X \to Y$ and $g' : Y \to X$ s.t. (1) $\forall x.\ px \leftrightarrow q(f'x)$ and $\forall y.\ qy \leftrightarrow p(g'y)$ and (2) $g'(f'x) = x$ and $f'(g'y) = y$.*

Proof. $f'x$ is defined as the unique y for which $(x, y) \in C_{\mathsf{S}(I_X x)}$ (which exists by Fact 50 (2) and is unique because $C_{\mathsf{S}(I_X x)}$ is a correpsondence sequence), and $g'y$ is symmetrically defined as the unique x for which $(x, y) \in C_{\mathsf{S}(I_Y y)}$.

(1) is immediate since C_n is a correspondence sequence. (2) is by case analysis whether $I_X x \leq I_Y y$ or vice versa. ◀

Theorem 52. *Let X and Y be discrete, enumerable types and $p : X \to \mathbb{P}$, $q : Y \to \mathbb{P}$. If $p \preceq_1 q$ and $q \preceq_1 p$, then there exist $f : X \to Y$ and $g : Y \to X$ s.t. for all $x : X$ and $y : Y$:*

$$px \leftrightarrow q(fx), \quad qy \leftrightarrow p(gy), \quad g(fx) = x, \quad f(gy) = y$$

Proof. See **??**. ◀

Corollary 53 (Computational Cantor-Bernstein)**.** *Let X and Y be discrete, enumerable types and $f : X \to Y$ and $g : Y \to X$ be injections. Then one can construct $f' : X \to Y$ and $g' : Y \to X$ s.t. for all $x : X$ and $y : Y$:*

$$g'(f'x) = x, \qquad f'(g'y) = y$$

10 Simple predicates

We now turn to Post's problem for $\preceq_m$, i.e. to finding an enumerable, undecidable predicate S s.t. $\mathcal{W} \not\preceq_m S$, i.e. S is m-incomplete. Post's observation was that the complement of all m-complete predicates are productive. It thus suffices to find an enumerable predicate with non-productive complement. A predicate $p : \mathbb{N} \to \mathbb{P}$ is productive if its non-enumerability is witnessed by a function f s.t. for every enumerable subpredicate $\mathcal{W}_c$ of p, p and $\mathcal{W}_c$ differ on fc.

$$\mathsf{productive}\, p := \exists f : \mathbb{N} \to X. \forall c.\ (\forall x.\ \mathcal{W}_c x \to px) \to p(fc) \wedge \neg \mathcal{W}_c(fc)$$

Lemma 54. *Productive predicates are not enumerable.*

Proof. Let p have a productive function f and be enumerable by φ_c via Fact 10. Then by the specification of f, $p(fc)$, i.e. $\mathcal{W}_c(fc)$, and $\neg\mathcal{W}_c(fc)$. Contradiction. ◀

Lemma 55. *$\overline{\mathcal{K}_0}$ is productive.*

Proof. Pick $\lambda n.n$. Let c be s.t. $\forall x.\ \mathcal{W}_c x \to \overline{\mathcal{K}_0} x$. Then $\mathcal{W}_c c \to \neg \mathcal{W}_c c$. Contradiction. ◀

Every productive predicate is Cantor-infinite and thus has an enumerable, Cantor-infinite subpredicate.

Lemma 56. *Every productive predicate is Cantor-infinite.*

Proof. From Fact 12 and since $x \in l$ is decidable, we obtain $c : \mathbb{L}\mathbb{N} \to \mathbb{N}$ s.t. $\mathcal{W}_{cl} x \leftrightarrow x \in l$ for every $l : \mathbb{L}\mathbb{N}$. Let p have a productive function f. We prove $\forall l : \mathbb{L}\mathbb{N}. \Sigma x.\ (\forall x_0 \in l. px_0) \to px \wedge x \notin l$, which suffices by a slight adaption of Lemma 42. Given $l : \mathbb{L}\mathbb{N}$, pick $x := f(cl)$. ◀

Corollary 57. *Every productive predicate has an enumerable, Cantor-infinite subpredicate.*

We can now show that the complement of m-complete predicates contains an enumerable, Cantor-infinite subpredicate since productiveness transports along many-one reductions:

Lemma 58. *Let $p \preceq_m q$. If p is productive, q is productive.*

Proof. Let f many-one reduce p to q, and let g be a productive function for p. Fact 12 yields k s.t. $\mathcal{W}_{(kc)} x \leftrightarrow \mathcal{W}_c(fx)$. Then $\lambda c.\ f(g(kc))$ is a productive function for q. ◀

Theorem 59. *Let p be m-complete. Then $\overline{p}$ has an enumerable, Cantor-infinite subpredicate.*

Proof. Since productiveness of $\overline{p}$ follows from $\overline{\mathcal{K}_0} \preceq_m \overline{p}$ and productiveness of $\overline{\mathcal{K}_0}$. ◀

In particular, then the complement of an m-complete predicate has a non-finite, enumerable subpredicate. Post's idea to find an enumerable, but undecidable, many-one *incomplete* predicate hinges on exactly this observation. We follow Post and define a predicate $p : X \to \mathbb{P}$ to be *simple* if it is enumerable, and its complement is both non-finite and does *not* contain a non-finite, enumerable predicate:

$$\mathsf{simple}\, p := \mathcal{E}p \wedge \neg\mathcal{X}\overline{p} \wedge \neg\exists q.\ (\forall x.\ qx \to \overline{p}x) \wedge \neg\mathcal{X}q \wedge \mathcal{E}q$$

Fact 60. *Complements of simple predicates are not enumerable.*

Corollary 61. *Simple predicates are undecidable.*

Theorem 62. *Simple predicates are m-incomplete.*

Lemma 63. *Given $p : \mathbb{N} \to \mathbb{P}$ and $p \times \mathbb{N} \preceq_1 p$, p is not simple.*

Proof. Let p be simple and f a one-one reduction from $p \times \mathbb{N}$ to p. We have to prove falsity, thus we can assume an element x_0 s.t. $\overline{p}x_0$ since $\overline{p}$ is non-finite by Fact 30. Now $\lambda x. \exists n. f(x_0, n) = x$ is a non-finite, enumerable subpredicate of $\overline{p}$. ◀

11 Post's simple predicate

We follow the presentation of Rogers [21, §8.1 Th. II] to construct the same simple predicate as Post [18, §5]. Recall that simple predicates are enumerable and that their complement is non-finite but may not contain a non-finite, enumerable subpredicate. The latter two properties will drive the construction, since either one is easy to establish on their own, but the combination needs care. Post's idea was to construct a predicate S containing an element from every non-finite (enumerable) predicate $\mathcal{W}_c$. Thus, $\overline{S}$ cannot have a non-finite, enumerable subpredicate. To ensure that $\overline{S}$ is still non-finite, S contains only a *unique* $x > 2c$ with $\mathcal{W}_c x$ for every large enough $\mathcal{W}_c$. The condition $x > 2c$ ensures that there are at least n elements less or equal $2n$ in $\overline{S}$, and thus $\overline{S}$ is non-finite.

The only technical difficulty in the definition of S is to obtain a unique x satisfying $x > 2c$ and $\mathcal{W}_c x$ for every large enough $\mathcal{W}_c$. Post ensures this by choosing the x enumerated first by φ_c (i.e. the x with the least index n $\varphi_c n = \mathsf{Some}\, x$) satisfying $x > 2c$. We abstract away from this property, and observe that any function mapping c to an $x > 2c$ does the job.

We fix a function $\psi : \forall c.\ (\exists x.\ \mathcal{W}_c x \land x > 2c) \to \mathbb{N}$ s.t. $\psi c H = x \to \mathcal{W}_c x \land x > 2c$ and $\psi c H_1 = \psi c H_2$, i.e. a proof-irrelevant choice function for the predicate $\lambda x.\ \mathcal{W}_c x \land x > 2c$. Such a choice function ψ can be constructed by using for instance $\mu_{\mathcal{E}}$ (Corollary 6).

We then define S as the image of ψ and prove that S is a simple predicate:

$$Sx := \exists c (H : \exists x.\ \mathcal{W}_c x \land x > 2c).\ \psi c H = x$$

Lemma 64. *S is enumerable.*

Proof. There is a strong enumerator $E : \mathbb{N} \to \mathbb{N}$ for $\lambda c. \exists x. \mathcal{W}_c x \land x > 2c$, i.e. we have $H{:}\forall n. \exists x. \mathcal{W}_{(En)} x \land x > 2 \cdot (En)$. Then, $\lambda n. \psi(En)(Hn)$ strongly enumerates S. ◀

Lemma 65. *$\overline{S}$ is non-finite.*

Proof. By Fact 36 it suffices to prove $\forall n. \neg\neg \exists L. |L| = n \land \# L \land \forall x \in L. \overline{S} x$. Given n, let $px := Sx \land x \leq 2n$. p is exhausted by $[0, \dots, 2n]$, thus it is not not listable. Since the claim is negative, we can assume some duplicate-free L_p listing p.

Now $|L_p| \leq n$: by decomposing Sx for every $x \in L_p$, we obtain L'_p with $\# L'_p$, $|L'_p| = |L_p|$, and $\forall c \in L'_p.\ c < n \land \exists H.\ \psi c H \in L_p$. Hence, $|L_p| \leq n$. Now the first n elements of $\mathsf{filter}(\lambda x.\ x \notin L_p)\, [0, \dots, 2n]$ form the wanted list. ◀

Lemma 66. *$\overline{S}$ contains no non-finite, enumerable subset.*

Proof. Let q be non-finite, contained in $\overline{S}$ and enumerated by φ_c via Fact 10. We derive a contradiction by showing $[0, \dots, 2c]$ to exhaust q: Assume qx. Then $\mathcal{W}_c x$ since φ_c enumerates q. We have to prove $x \in [0, \dots, 2c]$, which is stable. So let $x \notin [0, \dots, 2c]$ and derive a contradiction. Since $\mathcal{W}_c x \land x > 2c$ holds, there is in particular a proof $H : \exists x.\ \mathcal{W}_c x \land x > 2c$. By definition of ψ, we have $\mathcal{W}_c(\psi c H)$. By definition of φ_c, $q(\psi c H)$, and thus $\neg S(\psi c H)$, i.e. $\neg \exists c' H'. \psi c' H' = \psi c H$ – contradiction. ◀

Theorem 67. *S is a simple predicate.*

The construction of S settles Post's problem for $\preceq_m$:

Theorem 68. *There exists an enumerable, undecidable and m-incomplete predicate.*

Theorem 69. *$\preceq_1$ and $\preceq_m$ differ on enumerable, undecidable predicates.*

Proof. $S \times \mathbb{N} \preceq_m S$ holds, but $S \times \mathbb{N} \not\preceq_1 S$ by Lemma 63. ◀

12 Truth-table reducibility

Recall that p is many-one reducible to q if a decision for px can be computed from one instance of q, namely $q(fx)$. Truth-table reducibility generalises this intuition: A predicate p is truth-table reducible to q if a decision for px can be computed by evaluating a boolean formula with atoms of the form qy_i for finitely many queries y_i. Equivalently, boolean formulas with n inputs can also be expressed as truth-tables with 2^n rows, explaining the name "truth-table reducibility". We model the type of truth-tables by defining $\mathsf{truthtable}\colon \mathbb{T} := \mathbb{L}\mathbb{B}$. Given an assignment $l\colon \mathsf{truthtable}$ of length n, we use a canonical listing function $\mathsf{gen}\colon \mathbb{N}\to\mathbb{L}(\mathbb{L}\mathbb{B})$ $|\mathsf{gen}\, n| = 2^n$ and $\forall l\colon \mathbb{L}\mathbb{B}.\ l \in \mathsf{gen}\, n \leftrightarrow |l| = n$ to define

$$l \vDash T := \begin{cases} T[i] = \mathsf{Some}\,\mathsf{true} & \text{if } (\mathsf{gen}\,|l|)[i] = \mathsf{Some}\, l \\ \bot & \text{otherwise} \end{cases}$$

When convenient, we assume $l \vDash T\colon \mathbb{B}$ since $\lambda lT.\ l \vDash T$ is decidable. Any $f\colon \mathbb{L}\mathbb{B}\to\mathbb{B}$ can be converted into the truth-table $\mathsf{map}f(\mathsf{gen}\, n)$ with n inputs $l \vDash \mathsf{map}f(\mathsf{gen}\, n) \leftrightarrow fl = \mathsf{true}$ provided $|l| = n$. On paper, we abuse notation and treat any function $f\colon \mathbb{L}\mathbb{B}\to\mathbb{B}$ as truth-table.

A function $f\colon X\to\mathbb{L}Y \times \mathsf{truthtable}$ is a *truth-table reduction* from p to q if $\forall x.px \leftrightarrow l \vDash \pi_2(fx)$ for all lists l which pointwisely reflect q on the query list $\pi_1(fx)$. A predicate $p\colon X\to\mathbb{P}$ is *truth-table reducible* to a predicate $q\colon Y\to\mathbb{P}$ if there is a truth-table reduction:

$$p \preceq_{\mathsf{tt}} q := \exists f\colon X\to\mathbb{L}Y \times \mathsf{truthtable}.\forall x\ l.\ \mathsf{Forall}_2\ (\lambda yb.\ qy \leftrightarrow b = \mathsf{true})\ (\pi_1(fx))\ l \to px \leftrightarrow l \vDash \pi_2(fx)$$

Lemma 70. *1. Truth-table reducibility forms a pre-order.*
2. If $p \preceq_{\mathsf{tt}} q$ and q is decidable then p is decidable.
3. If $p \preceq_m q$ then $p \preceq_{\mathsf{tt}} q$.

Truth-table reducibility can employ negation and thus does not transport enumerability.

Fact 71. $\overline{p} \preceq_{\mathsf{tt}} p$

Fact 72. *Truth-table reducibility is an upper semi-lattice.*

To characterise truth-table reducibility in terms of one-one reduciblity, we introduce so-called truth-table cylinder, following Rogers [21, §8.4 Th. IX]. Given a predicate $p\colon X \to \mathbb{P}$, we define the predicate $p^{\mathsf{tt}}\colon \mathbb{L}X \times \mathsf{truthtable} \to \mathbb{P}$:

$$p^{\mathsf{tt}} := \lambda z.\ \forall l.\ \mathsf{Forall}_2\ (\lambda xb.\ px \leftrightarrow b = \mathsf{true})(\pi_1 z)\ l \to l \vdash \pi_2 z$$

The characterisation seems to be inherently non-constructive, we thus assume p to be stable.

Lemma 73. *Let $p\colon X\to\mathbb{P}$, $q\colon Y\to\mathbb{P}$, $x_0\colon X, y_0\colon Y$, and $g\colon \mathbb{L}X \times \mathsf{truthtable}\to Y$ be an injection. If p is stable, then $p \preceq_{\mathsf{tt}} q \leftrightarrow p^{\mathsf{tt}} \preceq_1 q^{\mathsf{tt}}$.*

Proof. We loosely follow the proof by Rogers [21, §8.4 Th. IX] and prove the following: (1) If p is stable then $p \preceq_1 p^{\mathsf{tt}}$. (2) $p^{\mathsf{tt}} \preceq_{\mathsf{tt}} p$. (3) Every reduction $p \preceq_{\mathsf{tt}} q$ can be given via an injective reduction function, provided an injection $f\colon X\to Y$ exists. (4) If p is stable, $f\colon X\to Y$ injective then $p \preceq_{\mathsf{tt}} q \to p \preceq_1 q^{\mathsf{tt}}$.

(1) and (2) are straightforward. For (3) assume $p \preceq_{\mathsf{tt}} q$ via a reduction function g and construct $g'x := (fx :: \pi_1(gx), \lambda\, b :: L.\ \pi_2(gx)L)$. g' is injective since f is injective. For (4), let $p \preceq_{\mathsf{tt}} q$ via an injection f by (3). Then f 1-reduces p to q^{tt}.

The claim from left to right follows using (1), (2), and (4), the direction from right to left needs (1) and (4). ◀

Corollary 74. *If $p, q\colon \mathbb{N}\to\mathbb{P}$ and p stable, $p \preceq_{\mathsf{tt}} q \leftrightarrow p^{\mathsf{tt}} \preceq_1 q^{\mathsf{tt}}$.*

13 A tt-complete simple predicate

We here present details for the construction and verification of the tt-complete simple predicate S^* (Lemma 80), based on the idea by Post [18, §8] and presentation by Rogers [21, §8.4 VIII].

Let $\mathcal{L}n := [2^n - 1, \dots, 2^{n+1} - 2]$, i.e. the number sequence starting at $2^n - 1$ of length 2^n). Then S^* is defined as

$$S^*x := Sx \vee \big(\exists\langle n, m\rangle.\mathcal{W}_n m \wedge x \in \mathcal{L}\langle n, m\rangle\big).$$

The non-finiteness proof of $\overline{S^*}$ will again crucially rely on $\forall\neg\neg\exists$ notions:

Lemma 75. $\forall n.\neg\neg\exists x \in [n, \dots 2n].\overline{S}x$.

Proof. Recall the Proof of Lemma 65 showing the non-non existence of some duplicate-free listing L of $(\lambda x.\overline{S}x \wedge x \leq 2n)$ s.t. and $|L| > n$. Lemma 28 applied to L and $[0, \dots, n-1]$ implies the claim. ◀

Fact 76. *$x \in \mathcal{L}n_1 \to x \in \mathcal{L}n_2 \to n_1 = n_2$ for all x, n_1, n_2.*

Lemma 77. *$\mathcal{W}$ and $\overline{\mathcal{W}}$ are generative.*

Proof. It suffices to show the predicates Cantor-infinite: $\mathbb{N} \hookrightarrow \mathcal{W}$ via the injection $\lambda n.(c_\top, n)$ and $\mathbb{N} \hookrightarrow \overline{\mathcal{W}}$ via $\lambda n.(c_\bot, n)$, where $c_\top$ and $c_\bot$ are codes obtained by Fact 13 s.t. $\forall x.\mathcal{W}_{c_\top} x \leftrightarrow \top$ and $\forall x.\mathcal{W}_{c_\bot} x \leftrightarrow \bot$. ◀

Lemma 78. *$\overline{S^*}$ is non-finite.*

Proof. It suffices to prove $\forall n.\neg\neg\exists x \geq n.\overline{S^*}x$ by Corollary 33. Let n be given. By Fact 31 and Lemma 77 there exists $\langle n_1, m_1\rangle > n$ with $\overline{\mathcal{W}}_{n_1} m_1$. The conclusion is negative, thus we obtain $x \in \mathcal{L}\langle n_1, m_1\rangle$ with $\overline{S}x$ and $x \geq n$ from Lemma 75.

We prove $\overline{S^*}x$. Let S^*x. Then either Sx, contradicting $\overline{S}x$, or $x \in \mathcal{L}\langle n_2, m_2\rangle$ for some $\mathcal{W}_{n_2} m_2$. Then, Fact 76 implies $\langle n_2, m_2\rangle = \langle n_1, m_1\rangle$, contradicting $\overline{\mathcal{W}}_{n_1} m_1$. ◀

Lemma 79. *S^* is simple.*

Proof. $\mathcal{E}S^*$ follows by closure properties, $\mathcal{E}S$ and $\mathcal{E}\mathcal{W}$. Non-finiteness of $\overline{S^*}$ is Lemma 78. Since $\forall x.\ \overline{S^*} \to \overline{S}$ and $\overline{S}$ was already shown to contain no non-finite, enumerable predicate (Lemma 66), also $\overline{S^*}$ contains no such predicate. ◀

Lemma 80. *S^* is tt-complete.*

Proof. $\mathcal{E}S^*$ since S^* is simple. We prove $\mathcal{W} \preceq_{\mathsf{tt}} S^*$ by

$$\lambda c x.\big(\mathcal{L}\langle c, x\rangle, \lambda L.\forall b \in L.b = \mathsf{true}\big)$$

Let L reflect S^* pointwise on $\mathcal{L}\langle c, x\rangle$. It suffices to show

$$\mathcal{W}_c x \leftrightarrow \forall y \in \mathcal{L}\langle c, x\rangle.\ S^* y.$$

The direction from left to right is immediate from the definition of S^*. For the converse direction, we obtain two cases:

1. $\forall y \in \mathcal{L}\langle c, x\rangle.\ Sy$: Contradiction to Lemma 75.
2. $\exists\langle c', x'\rangle.\mathcal{W}_{c'} x' \wedge y \in \mathcal{L}\langle c', x'\rangle$ for some $y \in \mathcal{L}\langle c, x\rangle$. Now, Fact 76 implies $\langle c, x\rangle = \langle c', x'\rangle$, and thus $\mathcal{W}_c x$. ◀

Theorem 81. *m- and tt-completeness do not coincide. In particular, $\preceq_m$ and $\preceq_{\mathsf{tt}}$ differ on enumerable, undecidable predicates.*

Proof. S^* is m-incomplete as a simple predicate, but tt-complete by Lemma 80. ◀

14 Hypersimple predicates

For settling Post's problem w.r.t. $\preceq_{\mathsf{tt}}$ in our synthetic setting we once more follow Rogers [21, §9.5] and introduce majorising functions: $f\colon \mathbb{N}\to\mathbb{N}$ *majorises* a predicate $p\colon \mathbb{N}\to\mathbb{P}$ if

$$\forall n.\neg\neg\exists l.\ \#l \wedge |l| = n \wedge \forall m \in l.pm \wedge m \leq fn.$$

Note that we slightly adapted the definition of majorising in order to be more suitable for constructive proofs. We immediately introduce a strengthening of the notion following Odifreddi [15]: A function $f\colon \mathbb{N}\to\mathbb{N}$ *exceeds* a predicate $p\colon \mathbb{N}\to\mathbb{P}$ if $\forall n.\neg\neg\exists i.\ n < i \leq fn.\ pi$.

Fact 82. *If f exceeds p, $\lambda n. f^n 0$ majorises p.*

Lemma 83. *If $\forall x.qx \to px$ and q is Cantor-infinite, there exists f exceeding p.*

Proof. Take $\lambda n.\ \pi_1(F[0,\dots,n])$, where F is obtained from the $\forall\Sigma$ formula in Lemma 42. ◀

Corollary 84. *Given* MP*, if there is no f majorising p, then $\overline{p}$ does not have a non-finite, enumerable subpredicate.*

Proof. Let no f majorise p and let q be a non-finite, enumerable subpredicate of p. By MP and Lemma 43, q is generative. By Corollary 45, q is Cantor-infinite. By Lemma 83 there is f exceeding p. By Fact 82, p is majorised – contradiction. ◀

The following is an adaption of Th. III.3.10 in [15], with

$$p \vDash (P,T) := \forall l.\ \mathsf{Forall}_2\,(\lambda xb.\ px \leftrightarrow b = \mathsf{true})\,P\,l \to l \vDash T.$$

Theorem 85. *If $\mathcal{K}_0 \preceq_{\mathsf{tt}} p$ there exists a function exceeding $\overline{p}$.*

Proof. Let g be a tt-reduction from $\mathcal{K}_0$ to p. By Fact 12 there is $c\colon \mathbb{L}\mathbb{N}\to\mathbb{N}$ s.t. $\forall lx.\ \mathcal{W}(cl)x \leftrightarrow \neg(\lambda x.x \notin l) \vDash gx$. Let $\mathsf{gen}_n : \mathbb{L}(\mathbb{L}\mathbb{N})$ contain exactly all duplicate-free lists l s.t. $\mathsf{max}\ l \leq n$. We define $a_{n,i} := c(\mathsf{gen}_n[i])$ for $i < l^n$ and $a_{n,i} := c[]$ otherwise. Then $\lambda n.\ 1 + \mathsf{max}\,[\pi_1(ga_{n,i}) \mid i < 2^n]$ exceeds $\overline{p}$. To show this, let n be given and assume

$$(*) : \forall j.\ n < j < \mathsf{max}\,[\pi_1(ga_{n,i}) \mid i < 2^n]\,.\ pj$$

We have to prove falsity. Note that $\neg\neg\exists.i < 2^n.\ \forall z.\ z \notin \mathsf{gen}_n[i] \leftrightarrow p^*z$ where $p^*z := (\neg\neg pz \wedge z \leq n) \vee z > n$. Since we have to prove falsity, we can assume such an i. Now by $(*)$ we have $\forall j.n < j \in \pi_1(ga_{n,i}) \to pj$ since $\pi_1(ga_{n,i}) < \mathsf{max}\,[\pi_1(ga_{n,i}) \mid i < 2^n]$ follows from $i < 2^n$. Since for $x \leq n$, $px \leftrightarrow p^*x$ by definition, we have $\forall x \in \pi_1(ga_{n,i}).\ px \leftrightarrow p^*x$. But now we obtain the following contradiction:

$$\begin{aligned} p \vDash ga_{n,i} &\leftrightarrow \mathcal{W}\,a_{n,i}\,a_{n,i} \leftrightarrow \mathcal{W}\,(c(\mathsf{gen}_n[i]))\,a_{n,i} \\ &\leftrightarrow \neg(\overline{\mathsf{gen}_n[i]} \vDash ga_{n,i}) \leftrightarrow \neg(p^* \vDash ga_{n,i}) \\ &\leftrightarrow \neg(p \vDash ga_{n,i}) \end{aligned}$$

□

A predicate p is *hypersimple* if it is enumerable and $\overline{p}$ is non-finite and not majorised:

$$\mathsf{hypersimple}\,(p\colon \mathbb{N}\to\mathbb{P}) := \mathcal{E}p \wedge \neg\mathcal{X}\overline{p} \wedge \neg\exists f.f \text{ majorises } \overline{p}$$

Theorem 86. *Hypersimple predicates are not* tt*-complete.*

▶ **Theorem 87.** *Given* MP*, hypersimple predicates are simple.*

15 Construction of a hypersimple predicate

We now construct and verify a hypersimple predicate, a result due to Post [18, §9]. We however follow Rogers [21, §8.1 Th. II], who presents a (more general) construction due to Dekker [4], defining a hypersimple predicate H_I for an arbitrary undecidable $I\colon \mathbb{N}\to\mathbb{P}$ with a strong, injective enumerator $E_I\colon \mathbb{N}\to\mathbb{N}$. By instantiating with e.g. $\mathcal{W}' := \lambda\langle c, x\rangle.\mathcal{W}_c x$, we obtain that $H_{\mathcal{W}'}$ is hypersimple, and thus enumerable, undecidable and tt-incomplete. The predicate $H_I\colon \mathbb{N}\to\mathbb{P}$ is defined as the so-called "deficiency predicate" of I:

$$H_I x := \exists x_0 > x.\ E_I x_0 < E_I x$$

Lemma 88. *$\overline{H_I}$ is non-finite.*

Proof. By Corollary 33 it suffices to prove $\forall x.\neg\neg\exists y \geq x.\overline{H_I} y$. We use complete induction on $E_I x$. Given $x\colon\mathbb{N}$ assume $(*)\colon \neg\exists y{\geq}x.\overline{H_I} y$. The claim is negative. Case analysis:

1. If $H_I x$, there exists $x_0 > x$ with $E_I x_0 < E_I x$. Hence, induction for x_0 yields $\neg\neg\exists y \geq x_0.\overline{H_I} y$ contradicting $(*)$.
2. If $\overline{H_I} x$ holds, x is an element as required. ◀

Lemma 89. *If f majorises $\overline{H_I}$, I is decidable.*

Proof. We prove that $g := \lambda x.x \in_{\mathbb{B}} \mathsf{map}\ E_I\ [0,\dots,f(\mathsf{S}\,x)]$ decides I if f majorises $\overline{H_I}$, i.e. $\forall x.\ Ix \leftrightarrow gx = \mathsf{true}$. The direction from right to left is easy, since E_I enumerates I. For the converse direction let $E_I n = x$ for some n. We show $n \in [0,\dots,f(\mathsf{S}\,x)]$, which is stable. Thus let $n \notin [0,\dots,f(\mathsf{S}\,x)]$, i.e. $n > f(\mathsf{S}\,x)$. Since f majorises $\overline{H_I}$ and the goal is falsity, we can assume l with $\#l$, $|l| = \mathsf{S}\,x$ and $\forall y \in l.\overline{H_I} y \wedge y \leq f(\mathsf{S}\,x)$. Let $m := \mathsf{max}(\mathsf{map} E_I\, l)$. We have

1. $m \geq x$, since $|\mathsf{map} E_I l| = |l| > x$ and $\#(\mathsf{map} E_I l)$.
2. $m = E_I m_0$ for some $m_0 \in l$ and therefore $m_0 \leq f(\mathsf{S}\,x)$ and $\overline{H_I} m_0$ by the properties of l.

We now prove $H_I m_0$ to obtain a contradiction. We have $n > f(\mathsf{S}\,x) \geq m_0$ and $E_I n = x \leq m = E_I m_0$. By the injectivity of E_I, $E_I n = E_I m_0$ implies $n = m_0$ (a contradiction), such that we have $E_I n < E_I m_0$ as required. ◀

Theorem 90. *There exists a hypersimple predicate.*

Proof. $\mathcal{E} H_I$ follows by (6) of Fact 93. ◀

Finally, we need to show $H_{\mathcal{W}}$ to be undecidable. Proving however that $H_{\mathcal{W}}$ is simple seems to require MP and also Rogers [21, §8.1 Th. II] gives a heavily classical Turing reduction from $\mathcal{W}$ to $H_{\mathcal{W}}$. We instead give a direct, constructive proof of $\mathcal{D} H_{\mathcal{W}} \to \mathcal{D}\mathcal{W}$, which implies the undecidability of $H_{\mathcal{W}}$.

Theorem 91. *$H_{\mathcal{W}}$ is undecidable.*

Proof. We prove $\mathcal{D} H_{\mathcal{W}} \to \mathcal{D}\mathcal{W}$ in Appendix C. ◀

Theorem 92. *There exists an enumerable, undecidable and* tt*-incomplete predicate.*

In general, a reducibility is a pre-order transporting decidability backwards. Since $\mathcal{W} \not\preceq_{\mathsf{tt}} H_{\mathcal{W}}$, but $\mathcal{D} H_{\mathcal{W}} \to \mathcal{D}\mathcal{W}$, $\preceq_{\mathsf{tt}}$ is not the most general reducibility notion.

16 Discussion

This paper provides formalised, mechanised, constructive, synthetic proofs of major well-known results in the area of reducibility theory. The synthetic perspective tremendously helped in all three other aspects: Synthetic proofs are easier to formalise, easier to mechanise, and easier to constructivise.

From our three main technical contributions, Myhill's isomorphism theorem is formalisable without any axioms. Thus it is a theorem about (constructive) logics as much as about computability theory. The theorem seems to be provable even in weaker logics: The operator $\mu_{\mathbb{N}}$ is not needed, and all recursions are of a simple structural form. For our second main contribution, i.e. Post's problem for $\preceq_m$, the construction of a simple predicate S only relies on a universal enumerator φ, whereas parametric universality is needed to prove m-incompleteness. Analogously, for Post's problem for $\preceq_{\mathsf{tt}}$, the construction and verification of a hypersimple predicate only need universality, and parametric universality is needed for the tt-incompleteness proof.

We would argue that a synthetic formalisation is an almost necessary pre-requisite for a mechanised proof of advanced computability-theoretic results: working in a model of computation in the phase of mechanisation where details are still unclear seems too tedious, since defining and verifying programs in a model of computation is time-consuming and should be delayed as much as possible. The concretisation of all synthetically developed results to a model of computation in a second step is then routine. Even in the mechanised proofs, no principal obstacles should occur. For Coq specifically, the automatic extraction of functions to a λ-calculus by Forster and Kunze [9] would simplify the task further.

Choosing CIC rather than other systems is crucial for projects in constructive reverse mathematics concerning synthetic computability theory: textbook proofs this paper is based on are often heavily classical, not only in the regard that different, constructively non-equivalent definitions of infinity are used interchangeably. It was thus conceivable that some results would depend on a classical logical axiom like LEM or the limited principle of omniscience LPO. Due to the separated universe of propositions CIC features, $\mathsf{CT} \wedge \mathsf{LEM}$ seems to be consistent, which is however inconsistent in e.g. HoTT [5]. However, in the end all our results do not require any classical assumptions. This means that the results can be transported to other type theories. Compatibility with classical logic is however still desirable both for possible investigations of classical synthetic computability theory as well as for purposes of constructive reverse mathematics.

For such future work, two questions seem most prominent: First, we would like to analyse whether our assumptions are minimal. It would be interesting to see whether MP is indeed necessary to prove that hypersimple predicates are simple and in general undecidable, or whether the assumption can be weakened. Secondly, with a naive synthetic definition of Turing reductions as partial functions $(A \to \mathbb{B}) \rightharpoonup (B \to \mathbb{B})$ transporting deciders, Post's problem becomes unsolvable, since $\mathcal{W}$ naively Turing-reduces to *any* undecidable predicate. Thus finding a faithful synthetic rendering of Turing reducibility allowing to capture Post's problem would be highly desirable.

References

1 Andrej Bauer. First steps in synthetic computability theory. *Electronic Notes in Theoretical Computer Science*, 155:5–31, 2006.

2 Douglas Bridges and Fred Richman. *Varieties of constructive mathematics*, volume 97. Cambridge University Press, 1987.

3 Nigel Cutland. *Computability: An introduction to recursive function theory*. Cambridge university press, 1980.

4 James C. E. Dekker. A theorem on hypersimple sets. *Proceedings of the American Mathematical Society*, 5:791–796, 1954.

5 Yannick Forster. Church's Thesis and Related Axioms in Coq's Type Theory. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*, volume 183 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:19, Dagstuhl, Germany, 2021. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. URL: `https://drops.dagstuhl.de/opus/volltexte/2021/13455`, `doi:10.4230/LIPIcs.CSL.2021.21`.

6 Yannick Forster. *Computability in Constructive Type Theory*. PhD thesis, Saarland University, 2021. URL: `https://ps.uni-saarland.de/~forster/thesis`.

7 Yannick Forster. Parametric Church's Thesis: Synthetic computability without choice. In Sergei Artemov and Anil Nerode, editors, *Logical Foundations of Computer Science*, pages 70–89, Cham, 2022. Springer International Publishing. `doi:10.1007/978-3-030-93100-1_6`.

8 Yannick Forster, Dominik Kirst, and Gert Smolka. On synthetic undecidability in Coq, with an application to the Entscheidungsproblem. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 38–51, 2019.

9 Yannick Forster and Fabian Kunze. A Certifying Extraction with Time Bounds from Coq to Call-By-Value Lambda Calculus. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:19, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: `http://drops.dagstuhl.de/opus/volltexte/2019/11072`, `doi:10.4230/LIPIcs.ITP.2019.17`.

10 Felix Jahn. *Synthetic One-One, Many-One, and Truth-Table Reducibility in Coq*. Bachelor's thesis, Saarland University, 2020.

11 Georg Kreisel. Mathematical logic. *Lectures in modern mathematics*, 3:95–195, 1965. `doi:10.2307/2315573`.

12 Bassel Mannaa and Thierry Coquand. The independence of markov's principle in type theory. *Logical Methods in Computer Science*, 13, 2017.

13 Andrei Andreevich Markov. The theory of algorithms. *Trudy Matematicheskogo Instituta Imeni VA Steklova*, 42:3–375, 1954.

14 John Myhill. Creative sets. 1957.

15 Piergiorgio Odifreddi. *Classical recursion theory: The theory of functions and sets of natural numbers*. Elsevier, 1992.

16 Christine Paulin-Mohring. Inductive definitions in the system Coq rules and properties. In *International Conference on Typed Lambda Calculi and Applications*, pages 328–345. Springer, 1993.

17 Pierre-Marie Pédrot and Nicolas Tabareau. Failure is not an option. In *European Symposium on Programming*, pages 245–271. Springer, 2018.

18 Emil L Post. Recursively enumerable sets of positive integers and their decision problems. *bulletin of the American Mathematical Society*, 50(5):284–316, 1944.

19 Pierre Pradic and Chad E. Brown. Cantor-bernstein implies excluded middle. *CoRR*, abs/1904.09193, 2019. URL: `http://arxiv.org/abs/1904.09193`, `arXiv:1904.09193`.

20 Fred Richman. Church's thesis without tears. *The Journal of symbolic logic*, 48(3):797–803, 1983.

21 Hartley Rogers. Theory of recursive functions and effective computability. 1987.

22 Robert I Soare. *Recursively enumerable sets and degrees: A study of computable functions and computably generated sets*. Springer Science & Business Media, 1999.

23 Matthieu Sozeau and Cyprien Mangin. Equations reloaded: High-level dependently-typed functional programming and proving in coq. *Proceedings of the ACM on Programming Languages*, 3(ICFP):1–29, 2019.

24 The Coq Development Team. The Coq Proof Assistant, version 8.11.0. `https://doi.org/10.5281/zenodo.3744225`, Jan 2020. `doi:10.5281/zenodo.3744225`.

25 The Coq std++ Team. An extended "standard library" for Coq. `https://gitlab.mpi-sws.org/iris/stdpp`, 2020.

26 Anne Sjerp Troelstra and Dirk van Dalen. Constructivism in mathematics. vol. i, volume 121 of. *Studies in Logic and the Foundations of Mathematics*, 26, 1988.

A Facts in synthetic computability

▶ **Fact 93.** *1. Decidable predicates are semi-decidable.*
2. Decidability is closed under (pointwise) $\wedge$, $\vee$, and $\neg$.
3. Listable predicates are decidable.
4. Semi-decidable predicates on enumerable types are enumerable.
5. Semi-decidability is closed under (pointwise) $\wedge$ and $\vee$.
6. The projections $\lambda y.\exists x.\, pxy$ and $\lambda x.\exists y.\, pxy$ of an enumerable predicate $p\colon X\to Y\to\mathbb{P}$ are enumerable.
7. Enumerable predicates on discrete types are semi-decidable.
8. Enumerability is closed under (pointwise) $\wedge$ and $\vee$.
9. Strongly enumerable predicates are enumerable.
10. Enumerable inhabited predicates are strongly enumerable.

▶ **Fact 94.** *1. $\mathbb{N}$ and $\mathbb{B}$ are discrete and enumerable.*
2. Both discrete and enumerable types are closed under pairs, sums, options, and lists.

A *retraction* is an injective function with explicit inverse. X is a retract of Y if there are $I\colon X\to Y$ and $R : Y \to \mathbb{O}X$ s.t. $R(Ix) = \mathsf{Some}\, x$ and $\forall xy.\, Ry = \mathsf{Some}\, x \to y = Ix$.

Fact 95. *A type is discrete and enumerable if and only if it is a retract of $\mathbb{N}$.*

This fact enables most of our results to only mention predicates $\mathbb{N}\to\mathbb{P}$, and then transport for free to predicates on infinite, enumerable, discrete types (such as $\mathbb{LB} \times \mathbb{ON}$).

B Cylindrification proofs

We characterise many-one reducibility in terms of one-one reducibility, after we have already done so for truth-table reducibility.

The proofs by Rogers [21, §7.6 Th. VIII] work via so-called cylindrification and we generalise it slightly to apply to base types other than $\mathbb{N}$. In our setting, a cylindrification of a predicate $p : X \to \mathbb{P}$ is $p \times Z$ for an inhabited type Z, and predicates of the form $p \times Z$ in general are called cylinders.

We fix two predicates $p\colon X\to\mathbb{P}$, $q\colon Y\to\mathbb{P}$ and $z\colon Z$.

Lemma 96. *Let $f\colon Y \times Z\to Z$ be an injection. Then $q \preceq_m p \leftrightarrow q \times Z \preceq_1 p \times Z$.*

Proof. We first prove:
1. $p \preceq_1 p \times Z$ given $z : Z$.
2. $p \times Z \preceq_m p$.
3. If $q \preceq_m p \times Z$ then $q \preceq_1 p \times Z$, provided an injection $g\colon Y \times X\to Z$.

The only interesting proof is (3): Let f reduce q to $p \times Z$. Then $\lambda y.(\pi_1(fy), g(y, \pi_1(fy)))$ is proves $q \preceq_1 p \times Z$. It is injective since g is injective and correct since f is correct.

Now, the direction from left to right follows from (1), (2), and (3). The converse direction from (1) and (2). ◀

Corollary 97. *Let $f\colon Y\to Z$ be injective. Then $p \preceq_1 p \times Z$ and for q s.t. $q \equiv_m p$ and $p \preceq_1 q$, $q \preceq_1 p \times Z$.*

C Undecidability of H_I

Finally, we need to show H_I to be undecidable. Therefore, we give a direct, constructive proof of $\mathcal{D}H_I \to \mathcal{D}I$. To do so, we define a predicate $\mathcal{B}\colon \mathbb{N}\to\mathbb{T}$ describing a certain kind of guardedness with guards in $\overline{H_I}$:

$$\mathcal{B}n := \Sigma x.E_I x \geq n \wedge \overline{H_I} x$$

Lemma 98. $\forall n x.\Sigma x' \geq x.E_I x' > n.$

Proof. By the injectivity of E_I using Lemma 25. ◀

Lemma 99. *Let H_I be decidable by some f. Then, $\forall n x'.(\mathcal{B}n) + (\Sigma x > x'.E_I x < n)$.*

Proof. Let H_I be decidable. Given n and x', we show

$$\forall n_0 \geq x'.E_I n_0 > n \to (\mathcal{B}n) + (\Sigma x > x'.E_I x < n)$$

by complete induction on $E_I n_0$:

Let $n_0 \geq x'$ with $E_I n_0 > n$. If $\overline{H_I} n_0$, $\mathcal{B}n$ holds and we are done. If $H_I n_0$ holds, we have $\exists n_1 > n_0.\ E_I n_1 < E_I n_0$. Applying $\mu_{\mathbb{N}}$ gives the stronger assumption $\Sigma n_1 > n_0.\ E_I n_1 < E_I n_0$. Then, induction for $E_I n_1$ implies the claim.

Finally, Lemma 98 yields $n_0 \geq x'$ as assumed in the claim. ◀

Lemma 100. *Let H_I be decidable by some f. Then, $\forall n.\mathcal{B}n$.*

Proof. Let H_I be decidable. Given n, we first show

$$\forall d.(\mathcal{B}n) + (\Sigma L.\#L \wedge |L| = d \wedge \forall x \in L.E_I x \leq n)$$

by induction on d:

The base case is immediate by picking $L := []$. In the step case, the inductive hypothesis either yields $\mathcal{B}n$ which shows the claim or a duplicate-free list L with $|L| = d$ and $\forall x \in L.E_I x \leq n$ and an element $x > \mathsf{max}\, L$. Now, Lemma 99 for $x' := \mathsf{max}\, L$ yields again either $\mathcal{B}n$ or an element $x > \mathsf{max}\, L$ with $E_I x \leq n$. Then, $(x :: L)$ is a list as required which.

For $d := 2 + n$ however, $\Sigma L.\#L \wedge |L| = 2 + n \wedge \forall x \in L.E_I x \leq n$ is contradictory: By the injectivity of E_I, $\mathsf{map}\, E_I\, L$ is duplicate-free, $|\mathsf{map}\, E_I\, L| = 2 + n > |[0, \ldots, n]$ but $\forall x \in \mathsf{map}\, E_I\, L.x \in [0, \ldots, n]$. Lemma 25 implies the contradiction. Hence, we have $\mathcal{B}n$. ◀

Lemma 101. $(\forall n.\mathcal{B}n) \to \mathcal{D}I.$

Proof. $\forall n.\mathcal{B}n$ yields $\forall n.\Sigma x.E_I x \geq n \wedge \forall x_0 > x.E_I x_0 > E_I x$ which implies $H : \forall n.\Sigma x.\forall x_0 > x.E_I x_0 > n$.

Then, $\lambda n.n \in_{\mathbb{B}} \mathsf{map}\, E_I\, [0, \ldots, \pi_1(Hn)]$ decides I. ◀

Corollary 102. $\mathcal{D}H_I \to \mathcal{D}I.$

Proof. Directly by Lemma 100 and Lemma 101. ◀

Corollary 103. *H_I is undecidable.*

It seems that this fully constructive proof of $\mathcal{D}H_I \to \mathcal{D}I$ can be turned into an again fully constructive Turing reduction from I to H_I, provided a reasonable notion of synthetic Turing reductions: Instead of applying $\mu_{\mathbb{N}}$ to the proof of $H_I n_0$ in Lemma 99, partial functions (that must be crucially present in Turing reductions) would allow to find $n_1 > n_0$ with $E_I n_1 < E_I n_0$ via an unbounded search.

D On the Coq Mechanisation

The Coq development accompanying the paper is sizeable due to the number of results covered, but due to the synthetic approach it is still concise.

Introducing the axioms for synthetic computability needs around 200 lines of code (LoC). The preparations regarding finiteness, Pigeonhole principles, and infinity, which are of independent interest in constructive mathematics and not only relevant for the present paper, take 500 LoC. Defining $\preceq_m$, $\preceq_1$, and $\preceq_{\mathsf{tt}}$ with the accompanying facts needs 700 LoC.

General results regarding simple and hypersimple predicates take 100 and 230 LoC respectively. The construction of S and S^* needs 700 LoC. The construction of H_I only needs 250 lines.

The development is relatively elementary: No advanced dependent types are needed. The Equations package [23] is used for a convenient definition of γ in the proof of Lemma 49, but a manual definition would have been equally possible.

Besides this, the `std++` library [25] is used for several convenient auxiliary functions regarding lists missing in Coq's standard library.