Oracle Machines and the Turing Jump in Synthetic Computability

Niklas Mück

March 7, 2022

1 Oracle Machines

It is well-known that the halting problem is undecidable, and that there are harder problems that still cannot be decided even when extending the model of computation with an oracle for the halting problem. In order to study the relative computability of decision problems, Turing [1] came up with the idea of oracle machines. An oracle machine is a Turing machine with an addition operation for querying a black-box solver for a given problem.

Having such a model of computation, one can introduce Turing reducibility $P \leq_T Q$ saying that a problem (set of numbers) P is computable relative to a problem Q if there exists an oracle machine with an oracle for Q that solves P. The first idea of this notion goes back to Turing [1] and was later developed by Post [2].

Because oracle machines only have a single additional operation compared to normal Turing machines, the Gödel encoding can be extended easily and one can construct a universal oracle machine (for a fixed oracle) very similarly to the universal Turing machine. Also the halting problem of oracle machines with an oracle for Q can be formulated as the set of all numbers encoding oracle machines that halt on their own encoding as an input. This set is also called the Turing jump [3] of Q, written Q'.

2 Synthetic Computability

We are working in constructive type theory implemented by the proof assistent Coq. Functions in Coq can only be defined by implementing them in the functional programming language Coq provides. Therefore it seems very natural to treat the type $\mathbb{N} \to \mathbb{N}$ as the type of computable functions. It allows us to build a whole theory on this idea without arguing in a concrete model of computation.

This approach was pioneered by Richman, Bridges, and Bauer [4, 5, 6] and advanced in constructive type theory by Forster et al. [7, 8]. It can be seen as "synthetic" meaning that the objects under investigation are defined axiomatically. It is dual to the "analytic" approach often used in textbooks where the objects under investigation are modeled in a greater system and afterwards interesting properties are derived.

Most computability theory textbooks first define a concrete model of computation e.g. Turing machines, then construct a Gödel encoding and argue that there is a universal Turing machine implementing a universal function. As a first key result they formulate the halting problem and show that it is not decidable.

In our synthetic setting we cannot do it in the same way as functions are not associated with their source code[1].
Instead we work with an enumerability of partial functions axiom (EPF) [4, 9]:

$$\mathsf{EPF} := \Sigma\theta : \mathbb{N} \to (\mathbb{N} \rightharpoonup \mathbb{N}).\ \forall f : \mathbb{N} \rightharpoonup \mathbb{N}.\ \exists c : \mathbb{N}.\ \theta_c \equiv f$$

We say that θ is an enumerator for elements of the type $\mathbb{N} \rightharpoonup \mathbb{N}$ which is therefore countable. In synthetic computability, θ plays a similar role as the universal function. Using this axiom we can formulate a problem $\mathcal{K}$ that is similar to the self-halting problem of Turing machines: $\mathcal{K}c := \exists v.\ \theta_c c \triangleright v$

3 Oracle Machines in Synthetic Computability

All functions $\mathbb{N} \to \mathbb{N}$ being computable becomes very handy when studying computable functions. However the existing setup does not extend that easily (by adding a single oracle operation) to oracle machines in order to study relative computability.

Forster [8] proposes a synthetic definition of Turing reducibility in his PhD thesis on which our work mainly builds on. It was conceived in joint work with Dominik Kirst and follows an idea by Bauer [10].
A **Turing reduction** consists of:

- a *functional relation transformer* $r : (Y \rightsquigarrow \mathbb{B}) \to (X \rightsquigarrow \mathbb{B})$ mapping functional relations $Y \rightsquigarrow \mathbb{B}$ (which can be seen as an oracle) to functional relations $X \rightsquigarrow \mathbb{B}$
- a *computable core* $r' : (Y \rightharpoonup \mathbb{B}) \to (X \rightharpoonup \mathbb{B})$ mapping partial functions to partial functions
- such that: $\forall R\ f.\ (\forall x\ y.\ R\ x\ y \leftrightarrow f\ x \triangleright y) \to (\forall x\ y.\ (r\ R)\ x\ y \leftrightarrow (r'\ f)\ x \triangleright y)$
- in addition it also needs to be continuous (see [8])

Functional relations are the non-computable counterpart to partial functions. Therefore the (potentially uncomputable) oracle is taken as a functional relation and is used in order to build another functional relation. The computable core makes sure that the only uncomputability comes from the oracle by requiring that the Turing reduction is computable by a partial function for all computable oracles. Continuity intuitively makes sure that the oracle is only queried finitely many times. We will study this property in detail later.

We take Forster's definition of Turing reducibility and derive a notion of **oracle machines for semi-decision** (naming the type $\mathbb{M}$) by changing the type to $r : (Y \rightsquigarrow \mathbb{B}) \to (X \rightsquigarrow \mathbb{1})$ and $r' : (Y \rightharpoonup \mathbb{B}) \to (X \rightharpoonup \mathbb{1})$. We call r the halting relation M_{halts} because choosing these types, halting means returning the unique value $\star$ of type $\mathbb{1}$. Then we define **oracle semi-decidability**:

$$\mathcal{S}_Q(P) := \exists M : \mathbb{M}.\ \forall x.\ x \in P \leftrightarrow M_{\mathsf{halts}}\ Q\ x$$

[1]this would even be inconsistent in classical set theory

We call a problem P semi-decidable with respect to an oracle Q if there is an oracle machine that halts – when passing an oracle for Q – on x if and only if x fulfills P.

4 Turing jump

The halting problem of oracle machines i.e. the Turing jump [3] of a problem Q is oracle semi-decidable relative to Q but its complement is not. We would like to find a synthetic definition for the Turing jump and show its properties.

In order to define the halting problem of oracle machines we need to relate oracle machines with natural numbers, their codes. We follow the definition of $\mathcal{K}$ and derive codes by enumerating oracle machines. As we would like to pass a natural number as an input and are only interested in whether the machine halts we reuse our definition of oracle machines for semi-decision.

Fact 1 gives that two oracle machines with the same computational core behave extensionally equal due to continuity reasons:

Fact 1. $\forall M\ M'\ R\ x.\ M_{\mathsf{core}} = M'_{\mathsf{core}} \to \neg M_{\mathsf{halts}}\ R\ x \to \neg M'_{\mathsf{halts}}\ R\ x.$

Therefore an enumerator $\zeta : \mathbb{N} \to ((\mathbb{N} \rightharpoonup \mathbb{B}) \to (\mathbb{N} \rightharpoonup \mathbb{1}))$ for computational cores is sufficient. Computational cores take a partial (computable) function and a natural number as an argument. Both argument types are countable so using the pairing function to combine them seems to be natural. Unfortunately we cannot computationally encode partial functions into natural numbers (there is only an $\exists c$ in EPF). So the type $(\mathbb{N} \rightharpoonup \mathbb{B}) \to (\mathbb{N} \rightharpoonup \mathbb{1})$ cannot be enumerated. We will fix this problem later by a tweak in the definition of oracle machines but skip the technical fix for now and assume an enumerator $\zeta : \mathbb{N} \to ((\mathbb{N} \rightharpoonup \mathbb{B}) \to (\mathbb{N} \rightharpoonup \mathbb{1}))$.

Now we can finally define the **Turing jump** of Q as follows[2]:

$$Q' := \{c \in \mathbb{N} \mid \exists M : \mathbb{M}.\ M_{\mathsf{core}} = \zeta c\ \wedge\ M_{\mathsf{halts}}\ Q\ c\}$$

Before we can prove that the Turing jump Q' is oracle semi-decidable with respect to Q but its complement is not, we need a prerequisite lemma. Namely that for each code c there is a oracle machine with a core of code c. Making such a construction continuous is a bit involved, therefore we skip the proof first.

Lemma 1. $\forall c.\ \exists M : \mathbb{M}.\ M_{\mathsf{core}} = \zeta c.$

Theorem 1. *Q' is oracle semi-decidable with respect to Q.*

Proof. Following the definition of oracle semi-decidability we need to find an oracle machine that halts – with oracle Q – on input n iff $n \in Q'$.
For the functional relation (relations on $\mathbb{1}$ are trivially functional) transformer we choose: $r\ R_o\ c\ \star := \exists M.\ M_{\mathsf{core}} = \zeta c\ \wedge\ M_{\mathsf{halts}}\ R_o\ c$
As computational core we choose: $r'\ f_o\ c := \zeta c\ f_o\ c$
Obviously r is equivalent to Q' when choosing an oracle for Q as R_o. But it remains to show that r' is a core of r. So given R and f such that f computes R we need to show: $\forall c.\ (\exists M.\ M_{\mathsf{core}} = \zeta c\ \wedge\ M_{\mathsf{halts}}\ R\ c) \leftrightarrow \zeta c\ f\ c \triangleright \star$

[2]we use set notation for predicates whenever it supports readability

$\rightarrow$ We assume there is a M such that $M_{\mathsf{core}} = \zeta c$ and $M_{\mathsf{halts}}\ R\ c$. In addition we already know that f computes R and can use the specification of the functional core to conclude that $\zeta c\ f\ c \rhd \star$.

$\leftarrow$ We assume $\zeta c\ f\ c \rhd \star$. Now Lemma 1 gives us an oracle machine M with $M_{\mathsf{core}} = \zeta c$. Again we know that f computes R and can use the specification of the functional core to conclude $M_{\mathsf{halts}}\ R\ c$. $\square$

Theorem 2. *$\overline{Q'}$ is **not** oracle semi-decidable with respect to Q.*

Proof. Assuming $\overline{Q'}$ would be oracle semi-decidable with respect to Q, we get an oracle machine M that halts (given an oracle for Q) on n iff $n \notin Q'$. Because ζ is surjective, there exists a c such that $\zeta c = M_{\mathsf{core}}$. Now it is enough to show that $c \notin Q' \leftrightarrow \neg M_{\mathsf{halts}}\ Q\ c$ as this contradicts our assumption.

$\rightarrow$ Assume $c \notin Q'$ and $M_{\mathsf{halts}}\ Q\ c$. We know that $M_{\mathsf{core}} = \zeta c$ and $M_{\mathsf{halts}}\ Q\ c$ by definition gives us $c \in Q'$ and leads to a contradiction.

$\leftarrow$ Assume $\neg M_{\mathsf{halts}}\ Q\ c$ and $c \in Q'$ this means there exists an oracle machine M' with functional core $\zeta c = M_{core}$ that does halt on c. However Fact 1 gives that if an oracle machine does not halt, then all oracle machines with equal core also does not halt which gives a contradiction. $\square$

References

[1] Alan Mathison Turing. Systems of logic based on ordinals. *Proceedings of the London mathematical society*, 2(1):161–228, 1939.

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

[3] Stephen C Kleene and Emil L Post. The upper semi-lattice of degrees of recursive unsolvability. *Annals of mathematics*, pages 379–407, 1954.

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

[5] Douglas Bridges, Fred Richman, et al. *Varieties of constructive mathematics*, volume 97. Cambridge University Press, 1987.

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

[7] 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 - CPP 2019*. ACM Press, 2019.

[8] Yannick Forster. *Computability in Constructive Type Theory.* PhD thesis, PhD thesis. Saarland University, 2021.: `https://ps.uni-saarland.de/~forster/thesis/phd-thesis-yforster-printblack.pdf`.

[9] 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, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference)*, volume 183 of *LIPIcs*, pages 21:1–21:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[10] Andrej Bauer. Synthetic mathematics with an excursion into computability theory (slide set). *University of Wisconsin Logic seminar*, 2020. `http://math.andrej.com/asset/data/madison-synthetic-computability-talk.pdf`.