

# An order-reversing embedding of Turing degrees into Arthur–Nimue–Merlin degrees

Jean Abou Samra (Eötvös Loránd University, Budapest, Hungary)

Preliminary joint work with David A. Madore (Télécom Paris, Palaiseau, France)

February 17th 2026

# Outline

- Introduction to Turing degrees
- A generalization: the Arthur–Nimue–Merlin game degrees
- Our result: Arthur–Nimue–Merlin game degrees embed Turing degrees order-reversingly
- Interpretation through realizability

# Computability

Concerned with the limits of computation in Turing-complete languages, e.g., Turing machines. Completely untyped: Gödel codes everywhere. Usually developed in classical logic. (But related to type theory through realizability, later).

Notion of partial computable function  $f : \mathbb{N} \dashrightarrow \mathbb{N}$ , of decidable problem  $P \subseteq \mathbb{N}$ .

How to prove that  $P$  is undecidable? Usual approach: through a **reduction**.

# Computability

Concerned with the limits of computation in Turing-complete languages, e.g., Turing machines. Completely untyped: Gödel codes everywhere. Usually developed in classical logic. (But related to type theory through realizability, later).

Notion of partial computable function  $f : \mathbb{N} \dashrightarrow \mathbb{N}$ , of decidable problem  $P \subseteq \mathbb{N}$ .

How to prove that  $P$  is undecidable? Usual approach: through a **reduction**.

**Many-to-one reduction** from  $P \subseteq \mathbb{N}$  to  $Q \subseteq \mathbb{N}$ : a computable function  $f : \mathbb{N} \rightarrow \mathbb{N}$  such that  $n \in P \Leftrightarrow f(n) \in Q$ . We write  $P \leq_m Q$ . If  $Q$  is decidable then  $P$  is decidable. So if  $P$  is undecidable,  $Q$  is undecidable.

Theoretical computer science starts with proving undecidability of the halting problem by diagonalization, then builds a network of undecidable problems by successive reductions (PCP, CFG universality, Hilbert's 10th problem, etc.).

## Turing reduction

Idea of the previous method: “Assume  $Q$  is decided by a program  $q$ . Then I build a program  $p$  to decide  $P$ , which uses  $q$ . But  $P$  is undecidable, contradiction.”

With many-to-one reduction, the program has a specific form:  $n \mapsto q(f(n))$ . With **Turing** reduction,  $p$  could use  $q$  arbitrarily.

## Turing reduction

Idea of the previous method: “Assume  $Q$  is decided by a program  $q$ . Then I build a program  $p$  to decide  $P$ , which uses  $q$ . But  $P$  is undecidable, contradiction.”

With many-to-one reduction, the program has a specific form:  $n \mapsto q(f(n))$ . With **Turing** reduction,  $p$  could use  $q$  arbitrarily.

We turn this into a positive notion using **oracles**. This defines **Turing reducibility**:  $P \leq_T Q$ . More generally  $f \leq_T g$  where  $f, g : \mathbb{N} \rightarrow \mathbb{N}$ .

Many ways to formalize this, e.g., a Turing machine with oracle is a Turing machine with a special tape. It can write a question  $n$  on this tape and trigger a special operation “call the oracle”. This instantly writes  $g(n)$  on the special tape and the machine continues. Then  $f \leq_T g$  when  $f$  can be computed by an oracle Turing machine with the oracle  $g$ .

## Comparing undecidable problems

When  $f \leq_T g$ , perhaps  $f$  is uncomputable, but  $g$  is *more* uncomputable. How do various problems compare in level of uncomputability?

Example: The halting problem  $h$  asks if a program halts without input; the universal halting problem  $u$  asks if a program halts on *all* inputs. We have  $h \leq_T u$  but  $u \not\leq_T h$ .

## Comparing undecidable problems

When  $f \leq_T g$ , perhaps  $f$  is uncomputable, but  $g$  is *more* uncomputable. How do various problems compare in level of uncomputability?

Example: The halting problem  $h$  asks if a program halts without input; the universal halting problem  $u$  asks if a program halts on *all* inputs. We have  $h \leq_T u$  but  $u \not\leq_T h$ .

The relation  $\leq_T$  is a preorder. We turn it into a partial order by quotienting:  $[f]_T = [g]_T$  when  $f \leq_T g$  and  $g \leq_T f$ . This gives the poset of **Turing degrees**.

Obvious questions:

- Is there a Turing degree between computable functions and the halting problem?
- Are the Turing degrees totally ordered?

Triggered a lot of research on the structure of the Turing degrees, which turns out to be very rich.

## Structure of the Turing degrees

- There exist infinitely many Turing degrees strictly between computable functions and the halting problem. However, no “natural” one has been found.

## Structure of the Turing degrees

- There exist infinitely many Turing degrees strictly between computable functions and the halting problem. However, no “natural” one has been found.
- Kleene-Post theorem: There exist two incomparable Turing degrees below the halting problem.

## Structure of the Turing degrees

- There exist infinitely many Turing degrees strictly between computable functions and the halting problem. However, no “natural” one has been found.
- Kleene-Post theorem: There exist two incomparable Turing degrees below the halting problem.
- In fact, *every* degree except the minimum degree (computable functions) is incomparable to some other degree.
- Moreover there exists an antichain of  $2^{\aleph_0}$  Turing degrees.

## Structure of the Turing degrees

- There exist infinitely many Turing degrees strictly between computable functions and the halting problem. However, no “natural” one has been found.
- Kleene-Post theorem: There exist two incomparable Turing degrees below the halting problem.
- In fact, *every* degree except the minimum degree (computable functions) is incomparable to some other degree.
- Moreover there exists an antichain of  $2^{\aleph_0}$  Turing degrees.
- The poset of Turing degrees has binary joins (easy), but not all binary meets.
- There exist minimal non-zero Turing degrees.
- Every countable poset embeds into the Turing degrees!

(See Wikipedia for more)

## Other kinds of degrees

- Many-to-one degrees: for  $\leq_m$
- Many more variants: enumeration degrees, Weihrauch degrees, truth-table degrees, Muchnik degrees, Medvedev degrees, ...
- In this work, we rename Turing degrees to T0 degrees and  $\leq_T$  to  $\leq_{T0}$ , and we consider T1, T2 and T3 degrees (our terminology), three successive extensions of T0 degrees

## T1 degrees: adding partiality

What does it mean to compute a partial function  $f : \mathbb{N} \dashrightarrow \mathbb{N}$  with a partial function  $g : \mathbb{N} \dashrightarrow \mathbb{N}$  as oracle?

1. We can forbid oracle queries where it is undefined (such queries make the computation fail).
2. We can also allow them, and allow several queries in parallel.

## T1 degrees: adding partiality

What does it mean to compute a partial function  $f : \mathbb{N} \dashrightarrow \mathbb{N}$  with a partial function  $g : \mathbb{N} \dashrightarrow \mathbb{N}$  as oracle?

1. We can forbid oracle queries where it is undefined (such queries make the computation fail).
2. We can also allow them, and allow several queries in parallel.
3. We can allow the program to do anything when called outside the range of  $f$ .
4. We can also require it not to terminate then.
5. Or we could even require it to return a special “undefined” value.

## T1 degrees: adding partiality

What does it mean to compute a partial function  $f : \mathbb{N} \dashrightarrow \mathbb{N}$  with a partial function  $g : \mathbb{N} \dashrightarrow \mathbb{N}$  as oracle?

1. We can forbid oracle queries where it is undefined (such queries make the computation fail).
2. We can also allow them, and allow several queries in parallel.
3. We can allow the program to do anything when called outside the range of  $f$ .
4. We can also require it not to terminate then.
5. Or we could even require it to return a special “undefined” value.

We choose 1 and 3. The oracle machine receives  $n$  and can assume  $f(n)$  is defined. It must only query values  $g(m)$  which are defined.

T1 degrees were studied by Kihara and Ng under the name “sub-Turing degrees”.

## T2 degrees: adding demonic non-determinism

We now extend the reducibility to partial functions  $\mathbb{N} \dashrightarrow \mathcal{P}(\mathbb{N})$ .

To use an oracle  $g$ , we ask a question  $n$ , the value  $g(n)$  must be defined, and we receive back *any* element of  $g(n)$ .

Conversely, to compute  $f$ , we receive an input  $n$ , we can assume  $f(n)$  is defined, and we need to output *any* element of  $f(n)$ .

## T2 degrees: adding demonic non-determinism

We now extend the reducibility to partial functions  $\mathbb{N} \dashrightarrow \mathcal{P}(\mathbb{N})$ .

To use an oracle  $g$ , we ask a question  $n$ , the value  $g(n)$  must be defined, and we receive back *any* element of  $g(n)$ .

Conversely, to compute  $f$ , we receive an input  $n$ , we can assume  $f(n)$  is defined, and we need to output *any* element of  $f(n)$ .

Unlike T0 (and T1?) degrees, there is a natural T2 degree strictly between computable functions and the halting problem: Let  $e$  be the  $n$ -th program, then  $f(n) = \{0\}$  if  $e$  returns true,  $f(n) = 1$  if  $e$  returns false,  $f(n) = \{0, 1\}$  if  $e$  does not terminate.

The T0 degrees above this T2 degree are known as the PA degrees and have been much studied. It is connected to LLPO in constructive math.

## T2 degrees reformulated via a game

Let  $f, g : \mathbb{N} \dashrightarrow \mathcal{P}(\mathbb{N})$ . We introduce a game between Arthur and Merlin. Arthur is the program, it must play with a computable strategy. Merlin is the oracle, it can use any strategy.

- A starting position is some  $n \in \mathbb{N}$  such that  $f(n)$  is defined. Arthur and Merlin can see  $n$ .
- Arthur and Merlin play alternately, until Arthur chooses to end the game. If Arthur never ends the game, Merlin wins.
- At each turn, Arthur can continue the game by choosing a query  $q$ . If  $g(q)$  is undefined, Merlin wins immediately.
- Otherwise, it is Merlin's turn. He chooses  $m \in g(q)$  (if  $g(q)$  is empty then Merlin is stuck and Arthur wins immediately). Arthur receives the value  $m$  and another turn begins.
- To end the game, Arthur declares a final value  $v$ . He wins if  $v \in f(n)$ , otherwise Merlin wins.

Arthur has a winning strategy iff  $f \leq_{T1} g$ .

## T3 oracles

In T2 oracles, we have *demonic* non-determinism: Merlin chooses his answer adversarially. In T3 oracles, we also have *angelic* non-determinism. We add a third player Nimue, who is allied to Arthur. She can play non-computably, but she cannot directly communicate with Arthur.

An oracle is now a total function  $g : \mathbb{N} \rightarrow \mathcal{P}(\mathcal{P}(\mathbb{N}))$ . To query the oracle, the program asks a question  $n \in \mathbb{N}$ . Nimue chooses  $P \in g(n)$  (or loses if  $g(n)$  is empty), which Merlin can see, then Merlin chooses  $r \in P$  (or loses if  $P$  is empty), which Nimue can see, and Arthur receives  $r$ .

## Reducibility between T3 oracles

At the T3 level, we only have the game to define reducibility. Let  $f, g : \mathbb{N} \rightarrow \mathcal{P}(\mathcal{P}(\mathbb{N}))$ .

- Arthur and Nimue are allied against Merlin.
- Nimue and Merlin can see everything that happens and can play with any strategies. Arthur only sees certain of the moves and his strategy must be computable.
- An initial configuration consists of  $n \in \mathbb{N}$ , which Arthur sees, and  $P \in f(n)$ , which only Nimue and Merlin see.
- At each turn, Arthur can decide to declare a final value  $v \in \mathbb{N}$ . Nimue and Arthur win if  $v \in P$ , otherwise Merlin wins. If Arthur never declares a final value, Merlin also wins.
- Arthur can also choose to query the oracle with a question  $q \in \mathbb{N}$ . If  $g(q)$  is empty then Nimue and Arthur lose. Otherwise, Nimue chooses  $Q \in g(q)$  which only Merlin sees. If  $Q$  is empty then Merlin loses. Otherwise, Merlin chooses  $r \in Q$  which both Nimue and Arthur can see and it is Arthur's turn again.

We put  $f \leq_{T3} g$  when Nimue and Arthur have a winning strategy.

## Examples

When the function  $g : \mathbb{N} \rightarrow \mathcal{P}(\mathcal{P}(\mathbb{N}))$  is constant, the oracle is *basic*. Arthur does not need to ask a question in this case (intuition: Nimue is able to read the question in Arthur's mind).

There is a greatest T3 degree  $\top$ , the **omnipotent degree**, represented by the basic oracle  $\{\emptyset\}$ . Nimue chooses  $\emptyset$ , and Merlin has no choice, so he loses immediately.

There is a second-greatest T3 degree  $\omega$ , the **omniscient degree**, represented by the basic oracle consisting of singleton subsets of  $\mathbb{N}$ . Nimue can communicate an answer to Arthur: she chooses  $\{n\}$ , Merlin must choose  $n$ , and Arthur declares the final value  $n$ .

## Examples (continued)

The **cofinite oracle** (first considered by Pitts) is the basic oracle consisting of the cofinite subsets of  $\mathbb{N}$ . Informally, the oracle gives arbitrary answers to questions “give an upper bound on something”.

**Theorem (van Oosten):** A function  $\mathbb{N} \rightarrow \mathbb{N}$  is computable with the cofinite oracle iff it is *hyperarithmetical*.

Hyperarithmetical functions are an important class of functions in classical computability, closely linked to descriptive set theory. Other characterizations of  $f$  being hyperarithmetical:

- $f$  is  $\Delta_1^1$
- $f \in L_{\omega_1^{\text{CK}}}$
- $f$  is computable by a “hyperarithmetical Turing machine”. These have an oracle for termination of other hyperarithmetical Turing machines.

## Our result

There are natural order embeddings from each level to the next:  $\mathcal{D}_{T_0} \hookrightarrow \mathcal{D}_{T_1} \hookrightarrow \mathcal{D}_{T_2} \hookrightarrow \mathcal{D}_{T_3}$ .

**Theorem:** We exhibit an order embedding  $\mathcal{D}_{T_0}^{\text{op}} \hookrightarrow \mathcal{D}_{T_3}$ . That is, we associate to each T0 degree  $d$  a T3 degree  $\text{co}(d)$ , such that  $d_0 \leq d_1 \Leftrightarrow \text{co}(d_0) \geq \text{co}(d_1)$ .

Sketch:

For  $f : \mathbb{N} \rightarrow \mathbb{N}$ , let  $H_{\text{true}}^f$  (resp.  $H_{\text{false}}^f$ ) be the set of programs with oracle  $f$  which return true (resp. false).

We define  $\text{co}([f])$  as the degree of the basic T3 oracle  $\{H_{\text{true}}^f, H_{\text{false}}^f\}$ .

Idea: the easier  $f$  is to compute, the more information Nimue can communicate to Arthur, so the more powerful  $\text{co}([f])$  is.

## Connection with realizability

Realizability is a way to build models of constructive mathematics which make precise the intuition that constructive proofs have algorithmic content.

The main such model is the **effective topos**. It is an elementary topos. We can also turn it into a model of extensional type theory with an impredicative extensional universe of strict propositions and type universes as allowed by our metatheory.

## A taste of realizability

Without defining the effective topos, here are some internally validated facts to get a feel of how living there is like:

- Every function  $\mathbb{N} \rightarrow \mathbb{N}$  is computable.
- Every function  $\mathbb{R} \rightarrow \mathbb{R}$  is continuous.
- There exists a subset  $X \subseteq \mathbb{N}$  which is neither finitely enumerated (surjection  $\{0, \dots, n-1\} \twoheadrightarrow X$ ) nor infinite (injection  $\mathbb{N} \hookrightarrow X$ ).
- There exists an uncountable subset  $X \subseteq \mathbb{N}$  (no surjection  $\mathbb{N} \twoheadrightarrow X$ ).
- There are countably many countable subsets of  $\mathbb{N}$ .
- Countable choice holds ( $\prod_{n \in \mathbb{N}} \|P(x)\| \rightarrow \|\prod_{n \in \mathbb{N}} P(x)\|$ ). Moreover, dependent choice holds (if for all  $x$  there exists  $y$  such that  $x \mathcal{R} y$ , then for all  $x_0$  there exists a sequence  $x_0 \mathcal{R} x_1 \mathcal{R} x_2 \mathcal{R} \dots$ ). Choice over  $\mathbb{N}^{\mathbb{N}}$  fails: no function code  $: \mathbb{N}^{\mathbb{N}} \rightarrow \mathbb{N}$ .
- Parallelism: assume  $\neg p_i \Rightarrow \neg\neg q \vee \neg q$  for  $i = 0, 1, 2$  and  $p_i \Rightarrow \neg p_j$  for  $i \neq j$ . Then  $\neg\neg q \vee \neg q$ .

## Lawvere-Tierney topologies

We use many modifier adverbs in mathematics. A proposition can be “locally true”, “true almost surely”, “true infinitely often”, etc.

Working constructively, let  $\Omega$  be the set of truth values.

A **Lawvere-Tierney topology**, or *modality* or *local operator*, is a function  $j : \Omega \rightarrow \Omega$  which we think of as an adverb to put on propositions:  $j(p)$  means that  $p$  is “ $j$ -true”. It must satisfy the requirements:

- $\forall pq : \Omega, (p \Rightarrow q) \Rightarrow (j(p) \Rightarrow j(q))$
- $\forall p : \Omega, p \Rightarrow j(p)$
- $\forall p : \Omega, j(j(p)) \Rightarrow j(p)$

In other words: it is a monad on the posetal category  $(\Omega, \Rightarrow)$ .

Lawvere-Tierney topologies on an elementary topos are in bijection with its subtoposes.

## Lawvere-Tierney topologies on the effective topos

The Arthur–Nimue–Merlin game was proposed by Kihara to give an understandable description of the Lawvere-Tierney topologies on the effective topos. He exhibited a natural order isomorphism between these and T3 degrees.

Examples:

- $\top$  is associated to  $j : p \mapsto \top$ . The subtopos is degenerate.
- $\omega$  is associated to  $j : p \mapsto \neg\neg p$ . The subtopos is **Set**.
- An oracle is T1 iff the subtopos is a realizability topos. Externally,  $j(p)$  is the set of programs which can witness  $p$  using the T1 oracle.

# Assemblies

Assemblies are a full subcategory of the effective topos.

An **assembly** is a set  $X$  with, for each  $x \in \mathbb{N}$ , a set of  $R(x) \subseteq \mathbb{N}$  of *realizers* of  $x$ . A morphism  $(X, R) \rightarrow (Y, S)$  is a map  $f : X \rightarrow Y$  such that there exists a Turing machine which maps realizers of  $x$  to realizers of  $f(x)$  for all  $x \in X$ .

- $\mathcal{2}$  is the assembly of booleans where 0 realizes true and 1 realizes false.
- $\nabla\mathcal{2}$  is the assembly of booleans where every code realizes every boolean (the “hidden booleans”).
- Given  $P, Q \subseteq \mathbb{N}$ , we define  $\mathcal{2}_{P,Q}$  as the assembly where elements of  $P$  realize true and elements of  $Q$  realize false.

## Relationship to the proof

Let  $P, Q \subseteq \mathbb{N}$ . We have inclusions  $2 \hookrightarrow 2_{P,Q} \hookrightarrow \nabla 2$ . Intuitively, the closer  $2_{P,Q}$  is to  $2$ , the further it is from  $\nabla 2$ .

To any subobject inclusion  $A \hookrightarrow B$  in an elementary topos, one can associate a Lawvere-Tierney topology  $j$ , which is the smallest for which  $A$  is  $j$ -dense in  $B$ , i.e.,  $\forall x \in B, j(x \in A)$ .

In this case, we take  $P := H_{\text{true}}^f$  and  $Q := H_{\text{false}}^f$ . Then the Lawvere-Tierney topology for  $f$  comes from the inclusion  $2 \hookrightarrow 2_{P,Q}$  and the one for  $\text{co}(f)$  from  $2_{P,Q} \hookrightarrow \nabla 2$ .

## Conclusion

- Our result is a step towards a better understanding of the structure of T3 degrees.
- T3 degrees deserve more attention: they seem to be a natural completion of the Turing degrees (hyperarithmetical degrees and PA degrees become just cones).
- The result and proof are written in terms of pure classical computability theory, but the concept of T3 degrees and the idea of the proof come from realizability. This is an application of constructive mathematics as a technical tool for classical mathematics.