Infinite Time Turing Machines for elementary proofs on recursive reals

Kenza Benjelloun[1] and Bruno Durand[2]

[1] Department of Mathematics and Geoscience University of Trieste, Italy, Laboratoire d'Informatique, Signaux et Systèmes de Sophia-Antipolis (I3S)de l'Université Côte d'Azur,Sophia-Antipolis, France
[2] LIRMM, Université de Montpellier, CNRS, Montpellier, France

Abstract. A famous theorem of recursion theory by Joseph Harrison published in 1967 [9] states the existence of sneaky recursive non-well-orders of which the well-ordered initial segment is longer than any recursive ordinal. This result is quite intriguing because it proves that some recursive orders are quite good at hiding their infinite descending chains. The proofs given by J. Harrison and others from the 60s reside typically in ATR_0 (second order logic on integers). With the development of transfinite computational models, namely Infinite Time Turing Machines in the 2000s, and the evolution of ordinal computability theory over the past 23 years, we seized the opportunity of giving this result a more elementary proof using ITTMs. From a reverse mathematics perspective, it is interesting to present a proof assuming as little logical requirements of the arithmetical hierarchy as possible and our proof is in ACA_0. Moreover, our work paves the way for an explicit construction of such orders.

1 Introduction

Our first objective, presented in section 2, is to give a precise explanation of how the stability of recursive ordinals under arithmetical oracles operates. In other terms, given an arithmetically defined ordinal, we illustrate a way to transform it into a *recursive* ordinal. We provide elementary proofs of old results about Kleene's $\mathcal{O}$, see for instance [10,7,11]; these will be useful to us later on. The theorem 1 is not new as a result. Our contribution resides in building a priority-argument-based construction proving it.

Our second objective, developped in section 3, focuses on sneaky recursive orders of which the well-ordered initial segment is longer than any recursive ordinal. The existence of such orders has been proved by Joe Harrison in [9].

Andreas Blass gave a clear explanation of this unexpected phenomenon in [2]:

> It is a known result of Spector and Gandy (Theorem III.3.5 in [12]) that a universal quantifier over hyperarithmetic reals amounts to an existential quantifier over arbitrary reals (without counting number quantifiers). So the set of codes of recursive linear orders with no hyperarithmetic descending sequence is included in Σ^1_1. But the set of codes of recursive well-orders is exactly the set Π^1_1. Therefore the two sets differ.

This proof, along with others present in the literature, uses second-order logic arguments. This makes for an elegant and short proof, but our ambition is to find a more *elementary* one. In fact, second-order logic assumes ATR_0, and what we call elementary is content with ACA_0 which is strictly weaker of an assumption. These classes are defined by central notions in reverse mathematics, and it is important to know where our proofs reside. ACA_0 is the proof system composed of Peano Arithmetics and the Arithmetical Comprehension Axiom. ATR_0 builds upon ACA_0 by adding Arithmetical Transfinite Recursion. In this system, the consistency of ACA_0 is proved, rendering it strictly stronger by Gödel's incompleteness theorem. For more details about these reverse mathematics proof systems, we refer the reader to the excellent chapter III of [6].

Our proof uses Infinite Time Turing Machines (ITTMs) as a computational model and our ordinal computation times are all recursive ($\in \omega_1^{\mathrm{CK}}$). Our original goal was to understand the limits of ITTM-computations but we realized that Harrison's result is *clearly visible* with ITTMs along the way.

The scheme of our proof is very simple : we design an ITTM that walks through recursive ordinals and we study its halting time. If Harrison theorem were false then this ITTM would halt in time exactly ω_1^{CK}, supremum of recursive ordinals, which is not possible. An additional effort has been made to remain as low as possible in the proof system we use, namely strictly within ACA_0, and here we use our results of the first part of the paper, in order to keep our proof "elementary" or in other terms "from below".

Definitions

We consider binary relations over the set of integer numbers; $a\mathcal{R}b$ means that a is in relation with b according to $\mathcal{R}$. The *support* of a relation is the set of integer elements that are in relation with at least one other element. If the support is ω we say that the relation is *total* and else it is *partial*. If the support is finite we say that the relation is *finite*, and in the sequel we consider only infinite relations. A binary relation can be encoded in a canonical way as a real number (we call *real number* a sequence of 0's and 1's indexed by ω) : we first fix a pairing function which is a bijection from ω^2 to ω denoted by $\langle\cdot,\cdot\rangle$. For any integer n, we get that $n = \langle a, b\rangle$ for some a and b. We define the real r which encodes the relation; its n^{th} bit is denoted by r_n such that $r_n = r_{\langle a,b\rangle} = 1 \iff a\mathcal{R}b$.

A real is said to be *recursive* if and only if there exists a Turing machine M such that M halts in finitely many steps and has the real r on its output tape.

A relation is *recursive* (resp. enumerable, $\Sigma_n^0, \dots$) if the real that represents it is recursive (resp. enumerable, $\Sigma_n^0, \dots$).

We say that two relations are *isomorphic* if we can renumber the elements of one order with the elements of the other one while preserving the relation. The renumbering is a bijection between both supports that preserves elements in the relation. Note that two isomorphic relations may be represented by different reals.

An *order* is a binary relation which is irreflexive, antisymetric and transitive. It can be total or partial as any relation. We denote our orders by the symbol $\prec$ and we recall that $\forall a\ a \not\prec a$ (irreflexivity).

A *linear*[3] *order* is an order such that all elements of its support are comparable. For a and b two distinct elements of the support we either have $a \prec b$ or $b \prec a$.

An order is a *well-order* if any subset of its support has a minimal element. Equivalently, we can define a well-order by one which contains no infinite descending chain. Some well known results include the *well-ordering theorem* and *Zorn's lemma*.

If an order is linear we can uniquely define its maximal *well-ordered initial segment*($w.o.i.s$).

We can interpret *countable ordinals* as isomorphism classes of well-ordered linear orders defined on ω. We say that an ordinal α is recursive (resp. enumerable, Σ^0_n,...) if at least one element of its isomorphism class is recursive (resp. enumerable, Σ^0_n,...).

Very often, we are interested in considering orders independently of the numbering of their elements (in our countable case the elements are integers). As for ordinals, we consider an order as an isomorphism class of orders (up to a permutation of the elements). The properties of orders which are usually studied (such as linearity and well-ordering defined above) are stable in the class. Thus when we say that an order is recursive it means – depending on the context – either that the precise order with this fixed numbering of elements is represented by a recursive real, or that there exists at least one other precise order in its isomorphism *class* which is recursive in the former sense. In the latter case, we say that there exists a re-numbering of elements such that the resulting order is encoded in a recursive real.

Prerequisites

The reader is assumed to be familiar with classical recursion theory as well as with the principles of infinitary computations, in this case Turing Machines which are allowed to compute for transfinitely many steps. We do not detail their behavior in this paper as they were introduced and thoroughly explained in detail in the pionneering paper[8]. We do also strongly recommend reading the reference book [3] which presents different infinitary analogues of famous models like infinite time register machines, α-ITTMs and others. We also point towards the papers [4,5] for learning more results and more algorithmic ways to use this ordinal computability model and understand its power.

[3] the term *total order* is also commonly used instead of linear order but we use the term *total order* when the support is total so we use a different term in order to avoid any confusion between the totality of the structure of the order and the totality of its support.

2 On recursive ordinals and stability

Proposition 1. *An enumerable infinite (countable) binary relation is total.*

According to the previous section, this means precisely that if there exists an enumerable real r representing a binary relation, then there exists an enumerable real r' with total support which represents the same binary relation – up to a recursive isomorphism. It appears that we can loose all isolated points by a recursive process. More precisely, we'll see in the proof that the enumeration program is recursively transformed

Proof: Let r be an enumerable real which represents a binary relation $\mathcal{R}$. Let's transform r into a real r' which encodes the restriction of $\mathcal{R}$ to its support. As we deal with infinite supports, in r' $\forall a \exists b\ a\mathcal{R}b$.

Since r is enumerable, we can use its enumeration machine to obtain an enumeration of pairs $\langle a, b\rangle$ such that $a\mathcal{R}b$. These pairs make up exactly the support of $\mathcal{R}$ as the union of all elements appearing in these pairs. At the beginning of the enumeration process, we obtain the first pair of comparable elements $\langle a, b\rangle$. Let's renumber a as 0 and b as 1. Once that's done, we put a 1 in our new real r' at the position $\langle 0, 1\rangle$ that represents the element $a\mathcal{R}b$ renumbered in r' as $0\mathcal{R}1$.

The correspondance between the old and the new numberings are stored in an array. When the enumerating machine outputs the second bit 1 of r, it has also computed its number $n = \langle a', b'\rangle$. The next step is to check if either a' or b' already have a numbering in the array. If they do, we keep it as is and skip to the next one. If not, we attribute to it the next available integer in the canonic order of ω $(0, 1, \ldots)$. Once that's done, we put a 1 in r' at the proper position: the renumbering of $\langle a', b'\rangle$. We keep repeating this process for all bits 1 of r.

In this way, the new real r' encodes the same binary relation $\mathcal{R}$ but is total and recursively obtained from the enumeration process of r. □

Beware that enumerable total relations are not always recursive. Indead there may be "unrelated" pairs of elements in it thus we cannot decide the relation. But if we switch to linear orders then this problem disappears and we get the following result.

Proposition 2. *Enumerable linear orders are recursive orders. Moreover, we can recursively transform the enumerating procedure of the considered order into the decision procedure of the recursive (linear) order.*

Proof: Let us consider an enumerable real r which encodes an *order* denoted by $\prec$. By our property on binary relations, we can transform r into a real r' which is recursive and encodes the same total order, meaning that it has the same comparable elements as r.

The order $\succ$ can be enumerated as follows : we enumerate $\prec$ up until we obtain $r'_{<a,b>} = 1$ with $a \neq b$, then output $r"_{<b,a>} = 1$. Thus $r"$ encodes the order $\succ$. Note that the enumeration machine of $r"$ is recursively obtained from the enumeration machine of r'.

We now prove that *if the order is linear*, the reals r' and $r"$ themselves are recursive. The following procedure decides them. If $n = \langle a, a\rangle$ for a given a, we output 0 to ensure antireflexivity. For other pairs $\langle a, b\rangle$ where $a \neq b$, in order to decide whether $r'_n = 0$ or $r'_n = 1$, it suffices to run in parallel the enumeration machines of r' and r", and we know that one of them halts because the orders are linear; we output a 0 or a 1 depending on which enumeration halts. This last transformation is also recursive, hence the result. □

In other terms, we have just proved that if a linear order is defined by a Σ^0_1 formula, then one can recursively obtain from it a Σ^0_0 (recursive) formula that defines a recursive total order isomorphic to the first one. We also proved that the property works for orders defined by Π^0_1 formulas.

If we focus on ordinals we get that if they are Σ^0_1 or Π^0_1, then they are recursive. To get higher stability results, we'll need to use all other properties of ordinals.

Theorem 1 (Spector). *If an ordinal is in the arithmetical hierarchy, then it is isomorphic to a recursive ordinal. Furthermore, the transformation of the formula that caracterizes the ordinal into this decision program (or machine) is recursive.*

Proof: First remark that it is sufficient to prove – with a relativizable proof – that if an ordinal is $0'$-computable then it is enumerable. Indeed, thanks to Prop.2, we would obtain that $0'$-computable ordinals are computable, and such would be all $0^{(n)}$-computable ordinals.

Now consider a $0'$-computable ordinal α *i.e.*, a $0'$-computable real r that represents an ordinal α. Thanks to Shoenfield lemma, r is computable at the limit. The rest of the proof is devoted to transforming the limit computation into an enumeration process for a real r' that represents an ordinal α' and proving that $\alpha \leqslant \alpha'$. We obtain an enumerable real r' because those bits declared 1 in r' never change back to 0.

The following proof, as was brought to our attention by a careful reader, is a result mentionned in (Bruno please help with)[1] as theorem 9.11, but its proof is not in this book, as it was left as an exercice. Its proof was futher not found in any of the classical recursion theory books the authors are aware of, but it was admitted as folklore that this proof exists using some priority argument within first-order logic. As we use 1 later on in section 3, we deliver hereafter our proposition of a formal proof.

We run the process of limit computation for more and more time on more and more inputs. In other terms, we imagine that we receive some information of the type $a \prec b$ time after time for different a's and b's. When we receive $a \prec b$ (bit 1 at position $\langle a, b\rangle$) we assume automatically that $b \not\prec a$ (bit 0 at position $\langle b, a\rangle$). After a finite number of steps, the value of the bit at position $\langle a, b\rangle$ and the bit at position $\langle b, a\rangle$ converge (and these convergences are of course compatible). We assume that all values are repeated infinitely often and oscillate at least once at the beginning.

Our process works step after step. A step begins each time a value $a \prec b$ is obtained, we process it and potentially we add a finite number of 1 in r' in order to obtain a finite order (order of finite support). At the end of each step, we take the transitive closure of what is written in the finite code in order to keep a finite (coherent as an order but maybe not linear) order coded in r'.

We construct a *table* t that considers those points enumerated in the support (e.g. points a and b when $a \prec b$ is delivered) and indicates the corresponding current points in r' namely $t(a)$ and $t(b)$. These $t(a)$ and $t(b)$ will be modified as time goes.

We use a counter f (integer) of free points (for r') which means that $x < f$ if and only if x is in the support of the finite order we construct, and if $x \geqslant f$ then x is in relation with no point (in r').

At any step, we say that an integer $p < f$ – that we interpret as a point in r' – is *active* or *inactive*. Active points are those points that have a correspondence in the table t (i.e. there exists an a such that $p = t(a)$).

For each integer p, that we interpret as a point in r', we construct a list of inactive points following p called its *tail*.

We consider priorities for integers a and b –which represent here points in r – in a very simple way : a is prioritary if $a < b$. Please note that the order relation we use is just the usual order $<$ on ω.

Properties of the algorithm. We manage so that points that enter the support are active and may be inactivated later (or not). Inactive points never become active again. Any inactive point belongs to a (single) tail. Only active points may have a tail. The tail of an (active) point p is totally ordered linearly and greater than p. Points in a tail are consecutive (tails do not interleave): an active point and its tail can be considered as a kind of block. For simplicity, we call "tail of a" the tail of $t(a)$.

We initialize our algorithm by the empty order which corresponds of course to $r' = 0 \ldots 0 \ldots$, the table t and the tails are empty and our counter f at 1. Indeed for simplicity, we consider a special point 0 that will be the minimum of our order in r'. When a point p enters our order we automatically add in r' that $0 \prec p$ and this information never changes.

Now let us explain how our process runs when $a \prec b$ is issued, *i.e.*, let us explain our *iteration step*.

If a or b (or both) are new, then we create a corresponding entry in the table with corresponding value f and then we increment f. In this case, we have no compatibility problem thus we add a 1 in r' at position $\langle t(a), t(b)\rangle$. But we also have to deal with the tail, so in case b is new, then we also add 1 in r' at position $\langle m, t(b)\rangle$ where m is the last element of the tail of a. We also add $0 \prec f$ in r' to keep 0 as a minimum. Remember that the iteration step will be completed by a transitive closure.

If a and b are both not new, we check wether we have (in r') $t(b) \prec t(a)$ and now we want that $t(a) \prec t(b)$, or that $t(b)$ and $t(a)$ were not compared before and are so now.

In the latter case, we just add $t(a) \prec t(b)$ ($m \prec t(b)$ where m is the last element of the tail of a) and nothing else: the iteration step is as usual completed by the transitive closure operation.

In the first case, we wish somehow to permute the relative position of $t(a)$ and $t(b)$ in the order r'. For this **we "move" the less prioritary** of a and b and keep the other one unchanged. We cannot really "move" points, since we want our order to be enumerable, so things are much more complex. We are going to use a new value for one of the points a and b and deactivate the old value.

Now we have to specify 2 cases : the first one is when a is prioritary on b and the second one is when b is prioritary on a. Let us start with the case of a prioritary. We use f as a new (free) representative for b and increment f. Let us first place f where it should be placed: just after the tail of a. We add $t(a) \prec f$ ($m \prec f$ where m is the last element of the tail of a) and for all active points q that were greater than $t(a)$ we add $f \prec q$. At the end of the loop we'll place f in the table t : $t(b) \leftarrow f$, but let us postpone this action after the deactivation of $t(b)$.

Deactivation of $t(b)$. First we select all immediate active predecessors of $t(b)$ namely $a_0, a_1 \ldots a_n$. Note that these points are not comparable and that there exists at least one of them thanks to our new minimum 0. We choose among them the predecessor with the least priority i.e. $a_i = \max\{a_0, a_1 \ldots a_n\}$. We then append $t(b)$ and its tail at the end of the tail of a_i (and delete the information about the tail of $t(b)$). For keeping the block structure, we decide that for $j \neq i$, $j \leqslant n$ $t(a_j) \prec t(a_i)$ and add the ad hoc relations in r'.

Now assume that b is prioritary on a. Then we create a new representative for a that we place just before b (as previously explained). We deactivate $t(a)$ as described above for $t(b)$, and then update the new value for $t(a)$ to the table t.

We then compute the transitive closure of our new order and proceed to the next input $c \prec d$.

Our process produces an enumerable real r'. Let us check its properties. First of all r' encodes an order because it is constructed as an extending finite order. If there was a problem with the order definition, then it would have appeared at some stage.

The order is linear. By construction, the order is linear by blocks. Now we can make our key remark : for each entry a in the table there exists a number of iteration of the process after which $t(a)$ never changes. This is due to the priority structure : $t(a)$ can be moved away only by a finite number of b's. This stabilization of the values of t is of great importance in our proof and let us denote by $t(a)_\infty$ the stabilized value.

Now it is clear that if in r we have $a \prec b$, which means that $a \prec b$ stabilizes as an output of the limit algorithm, then $t(a)_\infty \prec t(b)_\infty$ and reciprocally. Since r is linear, so is r'.

The order is well-founded. If we look precisely how tails are formed, new elements are appended. Hence at the limit, tails form segments of ordinal type ω and the well-foundation of r' comes directly from the well-foundation of r.

Thanks to the oscillation property assumed on the limit computation, we obtain a regular structure for r' : it consists of a minimum (our 0) followed by all $t(a)_\infty$ ordered as in r and separated by the tails. Those tails are of length exactly ω so the order type of the ordinal coded in r' is exactly $\omega.\alpha$. This construction relativizes thus the requirements stated at the beginning are proved and our theorem is proved. □

3 Ordinal type of a recursive real's *w.o.i.s*

Theorem 2. *[Harrison] There exists a recursive linear order of which the ordinal type of the well-ordered initial segment is exactly ω_1^{CK}.*

This theorem states that the *w.o.i.s* of a recursive real is not always recursive – somehow a surprising result. The theorem is usually proven by proving the existence of a special recursive real, a real encoding a linear order, which is shown to be prefixed by ω_1^{CK}. There have been several proofs of the existence of this linear order, formulated in second order logic over integers. Namely, the use of the powerful argument which is that $\Sigma_1^1 \neq \Pi_1^1$ which is typically in ATR_0. Our aim is to present a more elementary proof of this theorem in order ton understand better this (strange) recursive real thus we aim our proof to rely in ACA_0. We make use of the previously presented stability argument in addition to the use of Infinite Time Turing Machines as computation tools. Let us start with preliminary remarks, firts on ITTMs and then on recursive orders.

In their proof of the Count-Through theorem, J.D. Hamkins and A. Lewis describe what they call a *supertask*, which has to run several checks on a real given as input and decide whether it is a wellorder. Here we are not concerned with the ITTM-decidability of the class of wellorders, which is what is proven in the count-through theorem, but we are instead interested in the details of these *tasks* and more precisely in the clockable ordinals which are their exact halting times. For the need of our proof, we slightly rephrase and present here these *tasks* in the form of individual ITTM algorithms with an analysis of their respective halting times. For details on the Count-Through theorem, please refer to section 2.2 of the article [8].

Let us denote by μ_0 an ITTM which takes as input an integer n and tries to produce a real. This is done by running ω computations in parallell, where each one represents the work of n on one integer input. μ_0 halts in exactly ω steps. The first outcome would be the completion of the computations in finite time, where on each input k the Turing machine halted and answered some bit (the k-th bit of our real). And the second outcome would be that we detected that the TM did not halt on some input k.

Let us denote by μ_1 an ITTM which takes as input a real r and decides whether r is the encoding of a total linear order, namely whether r satisfies irreexivity, antisymetry and transitivity. In terms of halting time, μ_1 runs in no more than ω steps. This can be achieved, for example, by considering the indexes of the real by triplets and doing all three checks at the same time.

Let us also denote by μ_2 an ITTM which takes as input a real r encoding a total linear order $\prec$ and checks whether r is a well-ordering. This machine in turn halts in no more than $\alpha + \omega$ steps where α is the ordinal type of the *w.o.i.s* of r. To achieve this halting time, we use a technique for *emptying* the order from the bottom starting with 0. To select the elements to remove from the order, we compute by ω-*runs* all the first ω elements of the well-founded part of the order, if they exist. This is done by considering each element of the support in the integer order $(0, 1, 2, \cdots)$ and inserting each of them at its correct index (position) in the finite sequence of the previously considered integers (e.g. $d \prec a \prec c \prec \cdots \prec b$). We keep augmenting this sequence during ω steps, and we notice the stabilization of the index of the ω first well-ordered elements. In fact, they do stabilize because ω steps are enough to detect an index changing infinitely often, by flashing a flag for example. We iterate this process until we empty the order r from its *w.o.i.s.* If the remaining order is empty, then μ_2 halts in α steps and we read 1 on its output tape. If after this procedure the remaining order is not empty, the machine halts in $\alpha+\omega$ steps and we read 0 on its output tape.

Let us now remark that if an order is recursive, and a is an element of its support, the initial segment relative to a, $I_a = \{x,\ x \prec a\}$, is also recursive. Hence the *w.o.i.s* of r (some recursive linear order) may only contain recursive ordinals. Harrison theorem states that there exists a recursive linear order the *w.o.i.s* of which contains *all* recursive ordinals. Harrison states the existence of a recursive real having a *w.o.i.s* of ordinal type ω_1^{CK} In order to get it proved in ACA_0, we formulate it as : there exists a recursive real of which the *w.o.i.s* is of OT an ordinal greater than any recursive ordinal.

Proof: Let us combine our ITTMs μ_0, μ_1, μ_2 to obtain a *supertask* ITTMs which iterates through all Turing Machines n doing verifications as follows. First, we give n as input to μ_0, we halt upon it answering 0 and continue upon it answering 1. If we continue, it means that n has produced a (recursive) real r. We then give r as input to μ_1 to verify that it's a total linear order. (Note that we can parallelize these two procedures to get a ω step computation but this is not important). We thus continue, knowing that r is a total linear order. r is given as input to the machine μ_2, which computes during $\alpha+\omega$ steps, α being the $\mathcal{OT}$ of the *w.o.i.s* of r. Our ITTM supertask halts after the examination of Turing machines $0, 1, \ldots$, and its halting time, τ, is the sum of all computation times, τ_n, of each of the Turing Machines n sequentially considered by our procedure (the μ-computations).

These computation times form a set of ordinals $\{\tau_n, n \in \omega\}$ which is unbounded in ω_1^{CK}, since the latter is the supremum of all rectursive ordinals; if an ordinal β is recursive then there exists a Turing Machine number b that decides β and we have $\beta < \tau_b < \omega_1^{\mathrm{CK}}$. Subsequently, $\tau \geqslant \omega_1^{\mathrm{CK}}$.

Let us suppose now that all *w.o.i.s*-s α are recursive, contrary to our theorem's statement. The running time τ of our supertask ITTM is exactly ω_1^{CK} which is impossible, this ordinal being non clockable as an admissible – the theorem is proved.

This proof is based on the fact that no ITTM can halt in ω_1^{CK} by admissibility of the latter ordinal. Obviously, this argument is in second order logic, which does not serve our purpose of writing an ACA_0 proof. A transformation is thus necessary, let us explain now how it works.

First, let us recall that any ITTM computation is in $\Sigma_1(L_\alpha)$ where α is its halting time. Here it means that the considered $\Sigma_{n\in\omega}\tau_n$ is Σ_1-defined. Thus because of the recursive collapse over recursive ordinals of arithmetically defined ordinals (our Theorem 1) the sum is also recursive. This can also be expressed in more precise terms using Kleene's $\mathcal{O}$ notation [10]. Each of the τ_n can be defined in this notation since the machine n can be transformed into a notation for τ_n (we use here the recursive equivalence between recursive ordinals (as we defined them) and ordinals having a $\mathcal{O}$ notation – which is proved in one direction by standard computability arguments and in the other direction using Kleene's fixed point theorem). We thus get a notation for τ using partial sums $\Sigma_{n<N}\tau_n$ that contradicts it being greater than all recursive ordinals.

This argument does not use the admissibility of ω_1^{CK} and keeps our entire proof within ACA_0.

$\square$

References

1. C. J. Ash and J. Knight. *Computable Structures and the Hyperarithmetical Hierarchy.* Elsevier Science B.V., Sara Burgerhartstraat 25 P.O. Box 211, 1000 AE Amsterdam, The Netherlands, 2000.
2. Andreas Blass. Recursive non-well-orders. https://mathoverflow.net/questions/30633/sneaky-recursive-non-well-orders.
3. Merlin Carl. *Ordinal Computability, an Introduction to Infinitary Machines.* De Gruyter, Berlin, Boston, 2019.
4. Merlin Carl, Bruno Durand, Grégory Lafitte, and Sabrina Ouazzani. Admissibles in gaps. In Jarkko Kari, Florin Manea, and Ion Petre, editors, *Unveiling Dynamics and Complexity - 13th Conference on Computability in Europe, CiE 2017, Turku, Finland, June 12-16, 2017, Proceedings*, volume 10307 of *Lecture Notes in Computer Science*, pages 175–186. Springer, 2017.
5. Bruno Durand and Grégory Lafitte. An algorithmic approach to characterizations of admissibles. In Florin Manea, Barnaby Martin, Daniël Paulusma, and Giuseppe Primiero, editors, *Computing with Foresight and Industry - 15th Conference on Computability in Europe, CiE 2019, Durham, UK, July 15-19, 2019, Proceedings*, volume 11558 of *Lecture Notes in Computer Science*, pages 181–192. Springer, 2019.
6. Benoit Monin et Ludovic Patey. *Calculabilite : Aleatoire, Mathematiques a rebours et Hypercalculabilite.* 2021.
7. S. Feferman and C. Spector. Incompleteness along paths in progressions of theories. *The Journal of Symbolic Logic*, 27(4):383–390, 1963.
8. Joel D. Hamkins and Andy Lewis. Infinite time Turing machines. *Journal of Symbolic Logic*, 65(2):567–604, 2000.
9. Joseph Harrison. Recursive pseudo-well-orderings. *ransactions of the American Mathematical Society*, 131:526–543, 1968.
10. Stephen Cole Kleene. On notation for ordinal numbers. *Journal of Symbolic Logic*, 3:150 – 155, 1938.
11. Hartley Rogers. *Theory of Recursive Functions and Effective Computability.* MIT Press, Cambridge, MA, USA, 1987.
12. Gerald E. Sacks. *Higher RecursionTheory.* Springer-Verlag, Berlin Heidelberg, Germany, 1980.