Writability power of ITTMs: ordinals and constructible sets

Kenza Benjelloun, Bruno Durand, Grégory Lafitte

Writability power of ITTMs: ordinals and constructible sets

Kenza Benjelloun*
Bruno Durand†
Grégory Lafitte‡

February 25, 2024

Abstract

We present an optimal writability method for ordinals and Gödel's constructible sets L_α by infinite time Turing machines (ITTMs). For ordinals, we obtain that the time needed to write a writable ordinal is either ω, or the end of a clockability gap, or a limit of ends of such gaps. Our result is proved optimal and requires complex techniques for dealing with long gaps. As for Gödel's constructible sets L_α, we adapt our construction for writing ordinals for optimally writing L_α when α is ITTM-writable. We also obtain an optimal result (for closed enough α's): the time needed to compute L_α is the maximum between α and the ordinal time needed to write α.

1 Introduction

Ever since the birth of Infinite Time Turing Machines (ITTMs for short), many other infinitary analogues of computability and complexity themes have appeared as well (see the very good reference book [1]). Over the past two and a half decades, the study of ITTMs inevitably opened the discussion on the link between transfinite-time computation and set theoretical results. For a detailed study of this field, refer to the works [7, 11, 10, 12, 8, 2, 4]. ITTMs have proved to be useful models for both understanding ordinal computation and also for approaching the fine structure of the constructible hierarchy. Our contribution is also on this line.

Philip Welch in [13] section 3.1, presents a so-called "theory machine" from which he derives the "quick writing" of clockable ordinals and constructible sets – although not in optimal time. In the beginning of [14], more details on the functioning of this TM are given.

*Department of Mathematics and Geoscience, University of Trieste, Trieste, Italy
†LIRMM, Université de Montpellier, CNRS, Montpellier, France
‡LACL, Université de Paris XII, France, and Forum for Cultural Diplomacy

We introduce and prove the optimal bound for the writability time of any writable ordinal by ITTM. This result is the key point of the paper. Our proof also provides an alternative proof for the important theorem of Welch that states that any ITTM-clockable ordinal is also writable. We get optimality using a notion of stable writing. We then move on to explore the question of the ITTM-writability of L_α, the constructible sets of Gödel, which are also indexed by ordinals (here we restrict to writable ordinals). We present their writability time and prove optimality using our result on writable ordinals.

We start by introducing the main tools we use in our proofs of section 2 in the form of explanatory paragraphs and lemmas. After this exposition we enter the body of the proofs of the section, which consists in propositions 2, 3 and 4, followed by a synthesis of our main result in the form of a theorem derived from the precedent propositions. In section 3, we present a compilation of theorems which are the building blocks of our general scheme for computing a real code for constructible sets in an explicit way.

2 Stably writing ordinals

In our construction, we make frequent use of a variant of the universal machine for ITTMs. Note that it simulates all ITTMs in parallel without much loss of time : in any limit ordinal time, all ω ITTMs have been simulated [7, 2]. Then we enrich this simulation by observing those ω computations that are run in parallel. For instance, we can observe gaps in clockable ordinals. We explain how to construct an order on clockable ordinals "on-line". The term *on-line* means that a real encoding an ordinal α will be produced in time exactly α when α is a limit ordinal (it is also called real-time computation).

To understand the way this universal machine – which we will denote by $\mathcal{U}_{ol}$ – works, we can see the scratch tape and the output tape as a collection of ω scratch tapes and ω output tapes, each one pointed to by a cell n in the global tapes, which represent and simulate the computations of the infinite time Turing machine number n on a given input.

$\mathcal{U}_{ol}$ executes ω-*run* after ω-*run* of ω ITTM computations working in parallel (e.g. simulating n steps of the n first machines and then increasing n). At each ω-run, infinitely many machines may reach a halting state while infinitely many others do not. $\mathcal{U}_{ol}$ does not halt by design, it computes until we artificially put an end to it when the computation reaches a length of ordinal type some clockable ordinal.

To write reals using ITTMs, we need an "order-constructing" machine, denoted by $\mathcal{U}_{ck}$ (for *clockable*) inspired by the one defined in [4], which uses $\mathcal{U}_{ol}$ but adds some features: $\mathcal{U}_{ck}$ executes the code of each ITTM in the appropriate cell, and observes the first ITTM which halts on clockable time τ for each τ in order to build a well-order with these ITTM numbers. This order is isomorphic to the set of clockable ordinals : to each clockable ordinal we assign the number of the first ITTM which halts on time exactly this ordinal as observed by $\mathcal{U}_{ck}$. We also use this machine to observe the occurrence of gaps during its computa-

tions, allowing us to halt the program at our convenience and observe different results on its output tape.

We say that an ITTM writes a real r in a *stable* way if, during the writing procedure, the value of any cell of the output tape converges towards its final value; meaning that the sup rule is not applied to obtain the final output of the machine (no cofinal blinking in cells). This will obviously be the case in our first constructions since we write on the output tape but do not change the result later on : whenever a bit is printed out as 1 in the input tape, it is never changed. This stability will be much more difficult to obtain in the last construction aimed at long and complicated gaps (Proposition4) and will be discussed there.

This stability of our constructions is the key to proving the optimality of our main results on writing the ordinals and constructible sets in minimal ordinal time. We use the following lemma on ITTM-constructibility.

Lemma 1. *Let $f : \mathbb{R} \mapsto \mathbb{R}$ be a total recursive function. If a real r is stably writable in a limit time τ then $f(r)$ is also stably writable in time τ.*

Proof. We consider one of the Turing machines that compute the f function. We build an ITTM which takes r as input and a natural number n, and computes the n first bits of $f(r)$. The real r is constructed by another ITTM little by little and instead of writing it on the output tape, we write it in a work tape. We run the Turing machine for f on this partial information (so the computation of f might be wrong) and we output the result (if any) on the output tape of the ITTM at position n. This process is run on parallel with the rest of the ITTM computation. Remark that only a finite number of bits of r are needed to compute the n^{th} bit of $f(r)$. After some time, when the adequate bits of r have stabilized, our output stops fluctuating, it also stabilizes. The new ITTM halts in time τ because that's the time necessary to get all the information contained in the oracle r and the parallel run works correctly for limit ordinals. □

In this article, we often make implicit use of the result stating that any ordinal γ starting a gap in the clockable ordinals is admissible. To see the full proof, it appears as Theorem 50 in [13].

Let us now detail the writing of the ordinal ω_1^{CK} using the property that it starts the first gap in clockable ordinals. In what follows we denote by τ_α the end of a clockability gap that starts at α which is the ordinal segment $[\alpha, \tau_\alpha[$.

Proposition 1 (Writing of ω_1^{CK})**.** *There exists a machine $m_{\omega_1^{\mathrm{CK}}}$ that halts in time $\tau_{\omega_1^{\mathrm{CK}}} = \omega_1^{\mathrm{CK}} + \omega$ and* stably *writes a real code for ω_1^{CK}.*

This property can be obtained directly from [7] with a different construction, but we present our own detailed proof as it will be a stepping stone to understand the following constructions proving the more general results. Our statement is actually a direct corollary of our property 2 but we present a direct proof below. Presenting the details of the writing algorithm at many stages before the most general one is very important because we will use more elaborate tools as we go

along, in this manner the reader can see the progression of the writability and how the algorithmic power and complexity snowballs as the ordinals we consider get bigger.

Proof. Let's describe the algorithm for $m_{\omega_1^{\mathrm{CK}}}$. $\mathcal{U}_{ck}$ produces an order on ITTM numbers such as only one of the ITTMs halting in a certain clockable ordinal τ will have its number represented in the order, and that's the first one $\mathcal{U}_{ol}$ encounters. We use this machine to start building an order isomorphic to the set of clockable ordinals and we will halt this computation at the right ordinal time in order to have exactly a code for ω_1^{CK} on the output tape upon halting.

One way to detect this halting time is to launch, at the beginning of our computation, another ITTM called a *timer*, which halts after $\tau_{\omega_1^{\mathrm{CK}}} = \omega_1^{\mathrm{CK}} + \omega$ steps of computation, causing our machine to halt as well. We could use the technics from [7] for this but we choose to adapt $\mathcal{U}_{ol}$ which runs inside $\mathcal{U}_{ck}$ in order for it to detect a gap in clockable ordinals. A gap is detected exactly ω steps after the ordinal which began the gap since in our simulation we have to be sure that none of our ω simulated ITTMs has reached a halting state in that ordinal computing time.

At time $\tau_{\omega_1^{\mathrm{CK}}}$, what is written on the output tape is exactly the order representing ω_1^{CK} since it represents all clockable ordinals smaller than ω_1^{CK}.

The writing procedure is stable because as a new clockable ordinal is detected through the first machine which halts in that time, the corresponding bits are stably changed from 0 to 1. More precisely, when a new clockable ordinal is observed, ω bits are changed in the output to mark that it is greater than all preexisting ordinals in the order. □

In our precedent proof many variants are possible. For instance we could insert in the order ω elements as soon as a single halting machine is discovered in a ω-run, or just insert a single element (it would not harm the construction that much since gaps start on admissible ordinals which are closed enough [7]).

For the development of our proofs, we should be more precise on what happens before a gap and the following Lemma will be needed for that.

Lemma 2. *The ordinal type of the set of clockable ordinals occurring before a gap – which is began by an (admissible) ordinal α – is α itself. The ordinal type of the set of clockable ordinals occurring before a limit of gaps – which are began by (admissible) ordinals α_i – is the limit* $\sup \alpha_i$ *itself.*

This property can be proved in many different ways, and is a nice exercise to understand the closeness property of admissibles.

Proof. Let us first consider an ordinal α, the beginning of some isolated gap. This ordinal α is admissible. Consider β a clockable ordinal with no gap between β and α. The set γ, $\beta \leq \gamma < \alpha$ is a segment of clockable ordinals. Let us denote by δ its OT. We get that $\beta + \delta = \alpha$. But α is admissible, so by definition undecomposable into a finite sum of smaller ordinals. So $\delta = \alpha$ and the lemma is proved.

Now consider α being a limit of gaps. Since an isolated gap follows any gap, then α is also a growing limit of isolated gaps. We apply the result just above for those gaps and get the result. □

In the sequel we use Lemma 2 only for isolated gaps and for recursively inaccessible ordinals. The reader should keep in mind (from [2] for instance) that the length of an isolated gap is always ω: we can first clock an ordinal before the gap and then search for the next gap; identifying the gap requires ω steps. Limits of gaps are either non admissibles and thus clockable or admissibles and thus (recursively) inaccessible. The latter may start gaps of length ω (as does for instance the first recursively inaccessible) or much longer.

Proposition 2 (Writing time of beginnings of gaps)**.** *Let α be an admissible ordinal starting a gap. There exists a machine m_α which halts in time τ_α and stably writes a real code for α.*

Proof. Let's consider τ_α, the ordinal which closes the gap started at some ordinal α. Let n_{τ_α} be the number of an ITTM program which halts in time τ_α. Now let's run $\mathcal{U}_{ck}$ and n_{τ_α} at the same time. We make $\mathcal{U}_{ck}$ halt exactly as n_{τ_α} halts, and thus a code for all clockable ordinals smaller than α is written on the output tape. From Lemma 2 that's a code for α. The stability stems from the same reason as stated in the proof of Prop. 1. □

Proposition 3 (Writability time of ends of small gaps)**.** *Let α be an admissible ordinal starting a gap which doesn't contain other admissible ordinals. There exists a machine which halts in time τ_α and stably writes a real code for τ_α.*

Proof. Consider ordinals within the segment $[\alpha, \tau_\alpha]$ (and also those ordinals before the next gap). They are recursive in α since the gap contains no admissible ordinals: in order to go beyond recursion in α, we need to go beyond $\omega_1^{\mathrm{CK},\alpha}$: an admissible.

Hence we can apply Lemma 1 to them and transform our machine that writes the beginning of the gap (in our Prop. 2) into a machine that writes any of such ordinals, including τ_α. Still by Lemma 1, the writing is stable □

Proposition 4 (Writability time of ends of long gaps)**.** *Let α be an admissible ordinal starting a gap, $\alpha < \gamma_\infty$. There exists a machine which halts in time τ_α and stably writes a real code for τ_α.*

Here we deal with the more complex case when α starts a long gap. In particular, we recall that α is (recursively) inaccessible. The gap started in α can be arbitrarily large as we approach γ_∞ from below : the gap size is only bounded by γ_∞, see [2]. This means we have no algorithmic way of computing the size of the gap $[\alpha, \tau_\alpha[$ in advance, even if α is given as input, before reaching τ_α.

Remark that our result is *optimal* (as was the result of our previous propositions). If the writing time were less than τ_α then it would be a clockable ordinal γ with $\gamma < \alpha$. But the ordinal type of clockable ordinals in $[\gamma, \alpha]$ is exactly α

by admissibility. Thus we could program an ITTM that writes τ_α (or even α itself) and then count through this ordinal the clockable ordinals given by $\mathcal{U}_{ol}$ up to α. This machine would halt in exactly α steps. This would contradict the admissibility of α.

Proof. In the previous constructions, we used an order based on ITTMs' halting times. We handle ITTMs by their number (an integer) in the Gödel enumeration of ITTMs. We order these integers by the halting time of the corresponding ITTM when it runs on input 0. We need a unique integer representative for any clockable ordinal: we consider only the first ITTM that we discover halting in this time when running our universal machine.

Now that the gaps can be very long, their end is no more recursive in its starting point. We'll switch to oracle computation inside the gap, using the fact that the starting point of the gap (an admissible ordinal) is written on the tape (as in proof of proposition 2. This oracle computation will be used to find the next admissible inside the gap, etc. We'll explain later how to do this, but for now, please imagine that we keep trace of all admissibles and limits of admissibles, even when the limit is not admissible (we'll treat the same way all limits of admissibles, even if they are not recursively inaccessible).

We would like to represent both the number of the ITTM and the oracle it runs on – handled by an integer number. We thus encode both informations on a single integer number, by standard bijection between ω^2 and ω – denoted by $n = \langle a,b\rangle$ with $a = \Pi_1(n)$, $b = \Pi_2(n)$.

Note that at each end of gap we can erase all these extra ordinal representations that are only used to count through large gaps. Indeed after a gap we can come back to the counting of clockable ordinals and ignore ordinals in gaps below (our lemma 2).

Digression : if a reader would like to get a one-to-one correspondence between integers and writable ordinals we could keep the numbering obtained inside a gap. These ordinals would then be represented by triplets $\langle g, a, m\rangle$ where g is the name of the gap, a the rank number of the admissible or limit of admissibles that serves as oracle, and m the number of the ITTM working on oracle number a in the gap g that halts in this ordinal time. For g we can use the number of an ITTM that halts at the end of the gap – it provides a unique numbering for gaps. As this number is discovered after the gap, we could use a temporary special integer (e.g. 0) for the current gap and transform this integer into the correct value for g after the gap (it requires ω steps).

The first ordinals in the gap are the α-recursive ordinals. They are computed with ITTMs much in the same way that recursive ordinals are computed on oracle 0. Our aim is to order them with the associated information : "the ordinal denoted by the halting of the machine n on the β's admissible (or limit of admissibles) of the gap that ends with the halting of machine m". Let us discuss this more precisely. The machine n runs on an oracle. The first oracle chosen is α which is written on the tape. But we have to give a name to this α so that it is associated to this particular gap (and not a previous one). The name we chose for a gap is the end of the gap since it is clockable (by m). We

do not know m in advance, but m can be seen as a general parameter for the machine we construct. We need to prove that such a machine exists, there is no need (and no way) to construct if from below. The oracle α will be considered as the 0-th admissible of the gap. Now after the oracle α, when we arrive at $\omega_1^{\mathrm{CK},\alpha}$, we change our oracle and switch to $\omega_1^{\mathrm{CK},\alpha}$ in order to switch to a notation that means "the ordinal denoted by the halting of the machine n on the *first* admissible (or limit of admissibles) of the gap that ends with the halting of machine m". Then we switch to the *second* admissible, the ω-th, etc. For the sake of homogeneity, we switch our oracle as soon as an admissible or a limit of admissibles is found.

Now, it only remains to explain what integer notation we consider for β in the sentence "the ordinal denoted by the halting of the machine n on the β-th admissible (or limit of admissibles) of the gap that ends with the halting of machine m".

The first natural attempt would be to consider the number of β inside the order that we are currently constructing. This seems to be powerful enough since we can number more than α admissibles and thus overcome the problematic long gaps described in [2]. Unfortunately, this construction does not work : there are gaps that contain as many admissibles (in the sense of ordinal type) as the *end* of the gap. This can be proved easily using the techniques invented in [2] : the length of the gaps is unbounded in γ_∞ so if this construction had no fixed point, then we could observe this sequence of length of gaps (in terms of the order type of the admissibles they contain) and halt when the observed length is as long as the starting point of a gap. This would occur exactly in γ_∞ – impossible.

It remains to elucidate how in this numbering we can always get *fresh* numbers to use on every new oracle we need to insert in the construction of our order. This is easy for "relatively small" gaps, for instance if the number of admissibles inside the gap is bounded by α we could use the order for α that we use as first oracle. Unfortunately, the length of the gap, and even the number of admissibles inside cannot be bounded by any ITTM-computable function that take as input α and halts before the end on the gap. This is because those admissibles inside the gap can be detected and if such an ITTM-computable bound would exist, we would get an ITTM that halts after γ_∞ (again see [2]).

Thus, we have to renumber admissible ordinals in the gaps from time to time. This could be a problem: if we consider the sup of the ω first renumberings, because of the limsup rule, all the notations are messed up. We then have to reconstruct the order by a new computation and we explain how below. Please note that we can keep trace of these renumbering times and detect when we get to a limit of renumbering.

Let γ be a limit of such renumberings. We propose an ITTM algorithm to write γ stably in time $\gamma + \omega$. For this, we consider μ, one of the ITTMs that halts exactly at the end of the gap (note that for main part of the proof we can choose any machine that halts after the gap or any non-looping machine such as the universal ITTM). We call r_β its tape-configuration for limit time β. As μ is halting, all the r_β's are different (otherwise it would loop forever).

Now we get that each r_β can be uniquely described by a Σ_2-formula in L_γ. We call the number of this formula n_β, and this integer n_β will be used as a sort of index. We can compute n_β before β^+ (next admissible after β). For all cells of the machine that is 1 at a limit time, we can observe if it is obtained by stabilization or by the limsup operation.

Now we consider only times β when a renumbering was necessary (the set of β's is closed). If β is isolated, then we know how to renumber and we get a representation of β called R_β in time ω this easy renumbering takes time ω. Now we construct the triplet $(n_\beta, \mathrm{R}_\beta, \mathrm{S}_\beta)$ where S_β is an extra information on the stabilisation of cells in the configuration of μ called r_β. For each cell at 0, we store the list of n_γ (in incleasing order for the integer n_γ) such that the 0 has flashed 1 at least once between n_γ and n_β, the ordinal γ being in the list of renumbering as is β. Indeed we can observe if the cell has flashed 1 at least once since last renumbering and borrow the rest of the list from the triplet stored previously. For each cell at 1, we store only the bit information whether it was stabilising or obtained by limsup.

This triplet can be constructed stably as r_βs and by consequence the n_βs are all different (those reals are never erased). In other terms, the triplets we construct correspond to a bunch of ordinal representation stored in a non structured form, along with information on the limit behaviour of the non-looping ITTM μ.

Now let us consider γ which is limit in the β's (renumbering times). We can read the μ configuration r_γ and we know which 1's have stabilised before and which come from the limsup. We will use this precise information on r_γ and the triplets below to construct a sequence $(\beta_m)_{m\in\omega}$, cofinal in γ and the sum of their representation $\sum_{m\in\omega} \mathrm{R}_{\beta_m}$ which will be a representation of γ denoted by R_γ.

We select among the stored n_β's the triplet (order considered here is the normal order on integers considering n_β) such that the sequence S_β shows stabilisation for all stabilised 0's and 1's of r_γ and such that the m first oscillating bits (that will produce 1's) are correct (either at value 1 or at value 0 unstabilised since last chosen n_β for $m-1$). We can write this R_γ stably because when we remove a β because if it does not respect the conditions above on all the cells then we erase it forever.

Remark on the literature. Instead of counting time through halting machines with oracles, Philip Welch used an idea of 'Bigsum" machine. He then gets a procedure of renumbering analogous to ours necessary only on Σ_2-admissible ordinals. Then he constructs the set of true Σ_2 formulas in L_γ in real-time and uses lemma 1 of [6] to get an ITTM that extracts a real representation of γ from the set of true Σ_2 formulas in L_γ.

Now the proof is almost completed. We got a stable representation of all ordinals in the gap with delay at most ω with the help of the halting ITTM μ. Let us denote by λ the sup of rearrangements of the gap. If the end of the gap τ_α occurs at least ω steps after the last rearrangement ($\tau_\alpha > \lambda + \omega$) then an order for τ_α is (stably) written on the output tape when μ halts. But if $\tau_\alpha = \lambda$

then there is a small problem since the rearrangement requires ω steps.

The solution comes form an idea of Philip Welch in [13] (lemma 48 case 1). We observe the halting of μ. Instead of taking into account the whole limit configuration, we observe the first cell of each tape of μ. As they contain sufficient information for halting, one of them is at 1 for the first time and thus comes from cofinal blinking. Assume that the other one is stabilized at 0 (we can adapt μ to be in this case). Then we will stably build a cofinal sum $\sum_{m\in\omega} \mathrm{R}_{\beta_m}$ by selecting the β_m's online when the μ cell at 1 has flashed at least m times with the μ cell at 0 has been unchanged. This case is easier than the general case and occurs some times, for instance when the order type of admissible ordinals in a gap is equal to the end of the gap (and this happens).

To summarise our construction, we get an online counting of time with sometimes a delay ω, but when the delay is problematic to get the writing on time (end of the gap) then we use a simplified algorithm that uses the halting machine. □

Our result is somehow a modification the Welch's theorem which states that any clockable ordinal is writable : $\lambda_\infty = \gamma_\infty$ and of constructions presented by the same author in [13]. Our construction improves and simplifies (from our computability point of view) some of Welch's results.

Corrolary 1. *There exists a machine m that produces the real code of any writable ordinal in real-time.*

The term real-time in this corollary means that inside gaps, those codes are indeed produced with small delays $< \omega$, but cannot be showed (no ITTM can halt) before the end of the gaps – and that outside gaps such codes have been produced before. This result is not completely obvious from our proof that depends on a chosen μ that halts at the end of the gap but as mentionned earlier, it can be replaced by the universal ITTM (since it does not loop before λ_∞.

We can also get other results already proven in the literature concerning the size of the loops of non halting machines and relativise our construction to any oracle computation. Given a real number A, we can place it on the input tape at the begining of all computations and for instance get that $\lambda_\infty^{\mathrm{A}} = \gamma_\infty^{\mathrm{A}}$ We now state our main theorem.

Theorem 1 (Writability time of ordinals)**.** *If an ordinal is writable by ITTM, then its writability time is exactly the supremum of all ends of gaps that start before him (him included). If no gap exists before this ordinal, its writability time is exactly ω. The above writings can be made stable.*

The proof of this main theorem is nothing else than the combination of our Prop. 2, 3 and 4 using our stability lemma (lemma 1). It has been announced in [4] with a reference to [5], which unfortunately has not been published. The current paper is supposed to replace the latter.

In other terms, the writability times are exactly the ends of gaps and the limits of ends of gaps (and also ω for small enough ordinals. If an ordinal is not

in a gap then its writability time is exactly the sup of ends of gaps before, and if it is in a gap, then its writability time is exactly the end of that gap.

3 Writing Gödel's constructible universe

In this section, our focus is the ITTM-writability of L_α sets which contain the constructible sets of level α (see for instance Devlin's book [3]. Such sets do not necessarily represent an ordinal or even a real, but as our α's are countable, they can be encoded in a real. The considered constructible levels are those L_α for $\alpha \in \lambda_\infty$.

3.1 Results and theorems

We present the statements of our results of the section, and the proofs follow.

Theorem 2. *Given a code for an ordinal $\alpha \in \omega_1$ as input, L_α is ITTM-writable in time recursive in α (less than $\omega_1^{\mathrm{CK},\alpha}$).*

Two interesting theorems by Kœpke [9] on this topic are the ones which state that the set of codes of well-orders is computable by an ITRM, and second, that every Π^1_1 set $\mathrm{A} \subset \mathcal{P}(\omega)$ is ITRM-computable. In our article, we use ITTMs which are different models than ITRMs. Kœpke actually proves that any halting problem for ITRMs is ITTM-decidable with the same oracle (Th.3 in [9]).

We present our specific encoding of the sets L_α which enables us to write them into a real in optimal time using ITTMs. We do not start from Kœpke's theorems but from a theorem of evaluation from [1]. Constructible sets are generated by the constructibility operator Def, that we directly compute with ITTMs.

Theorem 3. *On input 0, the writing time of L_α, $\alpha \in \lambda_\infty$, is recursive in α's writing time.*

Theorem 1 gives us a writing algorithm for α in time τ_α which we combine with the result from Th. 2 to obtain our Th. 3.

Theorem 4. *On input 0, the writing time of L_α, when $\alpha \in \lambda_\infty$ is closed enough, is precisely $\max(\alpha,\tau_\alpha)$, where τ_α is the writing time of α.*

This theorem 4 is just a reinforcement of the previous one, given the construction. The hypothesis closed enough means that α should be closed in the computing time of Th 3. If this computing time were α^3 (we did not compute it precisely but it seems it is much less than this bound), then our bound concerns those α such that $\alpha = \sup_{\beta<\alpha} \beta^3$.

3.2 Construction of an L_α

In order to construct inductively $\mathrm{L}_{\alpha+1}$ from L_α we need to evaluate the validity of a formula.

Proposition 5 (Reformulation of Th. 2.3.28 in [1])**.** *There exists an ITTM $\mathcal{E}$ which computes the evaluation of formulas over L_α. More precisely $\mathcal{E}$ takes as input : L_α, a formula φ, and n integers k_i, where n is the number of the free variables x_i in ϕ. The machine $\mathcal{E}$ halts on all such inputs, outputing 1 if $\mathrm{L}_\alpha \models \varphi[k_i/x_i]$ and 0 otherwise, where k_i represents the i-th element pf L_α. The halting time of $\mathcal{E}$ is $< \omega_1^{\mathrm{CK},\alpha}$.*

Proof. A formula over L_α has a number in the enumeration of formulas, a certain number of variables and parameters. In order to evaluate it, we should be able to access the given code of each value for parameters. These codes are for elements in L_α, which is taken as input. So that code is accessible to $\mathcal{E}$. In addition, formulas are always the same countable sequences of symbols. What changes from a constructible level to the next is simply the sets in which the variables/parameters can be taken.

The computation of the formula itself is a very simple loop depending on its quantifier's alternation, and only relies on the fact that the machine can extract k_i from the code of L_α – this will be explained in the sequel. Remark that going through all k_i generates a loop of length only ω and not of length α since we do not have do go through any specific ordering of the elements. □

3.2.1 Finite levels of constructible sets

The first constructible set is $\mathrm{L}_0 = \emptyset$. Next levels are indexed by integers and $\forall n < \omega\ \mathrm{L}_n = \mathrm{V}_n$, meaning that finite constructible sets are exactly the 2^n finite sets added by the power set operation at each level. We can enumerate them by ITTM and hence write them all in time ω. We thus get a real that encodes L_ω ordered first by power set level and then by alphabetic order.

This real representing L_ω is the base case for our transfinite recursion with which we will be able to write other (infinite) levels of constructible sets. With this real, we can effectively access the encoding of different levels of constructible sets $< \omega$. Moreover, we can also obtain the k-th element of the n-th level in finite time by and ITTM that gets L_ω (encoded in a real as explained above) as input.

3.2.2 Successor levels : an explicit encoding

We suppose that we dispose of the encoding r_{L_α} of the constructible set of level some $\alpha > \omega$. We give the algorithm which writes the real, $r_{\mathrm{L}_{\alpha+1}}$, of the successor of α

$$\mathrm{L}_{\alpha+1} = \mathrm{Def}\,(\mathrm{L}_\alpha)$$

This means that $\mathrm{L}_{\alpha+1}$ is precisely the set of all elements (which are also sets) $a \in \mathrm{L}_\alpha$ which are produced by a formula such that

$$\exists n\ a = \{x \in \mathrm{L}_\alpha |\ \models_{\mathrm{L}_\alpha} \varphi_n(\dot{x})\}$$

The expression $\varphi_n(\dot{x})$ represents the resulting set obtained by applying the formula number n of our language to the constant symbol of the set x. As the

constructible set that we are currently building is $\mathrm{L}_{\alpha+1}$, let's denote its elements by a_i. To point out elements of the predecessor level L_α, we will use variables b_i. An element a_n of $\mathrm{L}_{\alpha+1}$ has its elements defined by a formula φ_n and has the form

$$a_n = \{b_0, b_1, b_2...\} \subseteq \mathrm{L}_\alpha$$

This set is a subset of L_α which can be represented by a real, and its elements are exactly those elements of L_α which satisfy the associated formula φ_n.

Let's consider the grid ω^2. We use it to encode our successor set $\mathrm{L}_{\alpha+1}$.

On the horizontal axis are represented the ω formulas of $\mathscr{L}_{\mathrm{L}\alpha}$ which each have a number. Each set produced by a formula will be represented by a real encoded in the column numbered by its index.

As for the lines, for each line of index m, there will be a 1 in $\langle n,m\rangle$ if and only if the set b_m (of L_α) is an element of a_n, and 0 otherwise. The answer to this question is given by our Prop 5. What needs to be done at present is to enumerate and evaluate all possible formulas. To do so, we may use theorem 5. In this way, we obtain an encoding in ω^2 bits of our desired set $\mathrm{L}_{\alpha+1}$, and as it is clear that ω^2 is isomorphic to ω, we can in turn obtain a real encoding for our desired set, which was the goal all along. To decode it, we must only know of the fixed bijection we used to encode the ω^2 bits into ω.

3.2.3 Successor levels : encoding with references

Let α be some ordinal. We would like to build the encoding of $\mathrm{L}_{\alpha+1}$ implicitly by referring to the code of L_α without rewriting the entire code for each constructible element. For doing this, we use the construction of our Prop. 4 which constructs incrementally a representation for ordinals. In this representation, we use ω bits in order to store L_α. We get these ω bits very easily : our order representing a level α is based on the halting of ITTMs at that time with a given oracle. But get ω machines that halt at the same time. In our construction, we chose the first discovered machine (with our machine $\mathcal{U}_{ck}$). In the current construction, we use all these numbers to store the ω bits of L_α.

3.2.4 Limit levels

Let λ be a limit ordinal. We aim in this section to exhibit a technique of encoding for the constructible L_λ into an ITTM-computable real, given r_λ.

Each ordinal $\alpha \prec \lambda$ has a number in r_λ. We can use these very numbers to vertically store the reals of each constructible level L_α. We construct in this way a code of length ω^2 for L_λ, where in each column number n is stored the corresponding code of the level L_α where α is the ordinal number n in r_λ. To obtain a real for a limit level constructible set using this technique, it will only remain to project its result using $\langle\cdot,\cdot\rangle$.

3.2.5 Improved encoding of limit levels

In our technique presented above, There is nothing specific to do in limit levels, excepted to keep a correct ordering. But for the sake of homogeneity, we would like to indicate what is the element number n of such a L_α. It should be the m's element of a L_β for a $\beta < \alpha$ and this β has an integer number in α. So we can solve this indirection and give directly the proper reference to the n's element (as a pointer resolution).

Proof of Th. 3. We have just explained how the L_α's are constructed and what is their encoding. Both go together since the encoding relies on the construction itself in a sophisticated reference system. The evaluation of a formula is given in our Prop. 5. We can now combine these results and get Th. 3. The time bound stated is rough : we just stated that it is recursive in α which is completely clear, given the structure of the described algorithm. □

The structure of the construction algorithm of embedded controlled loops gives for sure a polynomial time bound (in α). We think it is as simple of α^3 or even α^2 which is of little importance.

3.3 Time of writability of constructible sets

Proof of Th. 4. Let $\alpha \in \lambda_\infty$ be an ordinal closed by the complexity function of the construction discussed above. We characterize its ITTM-writability time by the ordinal $\max(\tau_\alpha,\alpha)$. Beware that when α is not clockable then $\alpha < \tau_\alpha$ but when α is clockable then $\tau_\alpha \leq \alpha$.

In the first case, α is in a gap. Thus if we denote by α_0 the beginning of the gap, we could construct simultaneously α_0 and L_{α_0} at time α_0 (α_0 being admissible is closed enough). Please note that this is a construction "at the limit" and we cannot halt at this point. The same property holds for β being the sup of all admissibles of the gap that may occur before α. Then the rest relies on the closure property of α.

Now if α is clockable, $\tau_\alpha \leq \alpha$. Now remark that α is recursive in β (still defined as the sup of all admissibles before α). The set L_β is constructed "at the limit" in time β and thus we can use our theorem 3 to get the result. □

References

[1] Merlin Carl. *Ordinal Computability, an Introduction to Infinitary Machines*. De Gruyter, Berlin, Boston, 2019.

[2] 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.

[3] Keith J. Devlin. *Constructibility*. Perspectives in Logic. Cambridge University Press, 2017.

[4] 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.

[5] Bruno Durand and Grégory Lafitte. A constructive swissknife for infinite time Turing machines. 2016.

[6] Sy-David Friedman and Philip D. Welch. Two observations regarding infinite time turing machines. 2011.

[7] Joel D. Hamkins and Andy Lewis. Infinite time Turing machines. *Journal of Symbolic Logic*, 65(2):567–604, 2000.

[8] Peter Koepke. Turing computations on ordinals. *Bull. Symb. Log.*, 11(3):377–397, 2005.

[9] Peter Koepke and Russell Miller. An enhanced theory of infinite time register machines. 5028:306–315, 06 2008.

[10] Philip Welch. Eventually infinite time Turing degrees: infinite time decidable reals. *Journal for Symbolic Logic*, 65(3):1193–1203, 2000.

[11] Philip Welch. The length of infinite time Turing machine computations. *Bulletin of the London Mathematical Society*, 32(2):129–136, 2000.

[12] Philip Welch. The transfinite action of 1 tape Turing machines. In *New Computational Paradigms, Proceedings of the First international conference on Computability in Europe, CiE 2005, June 8-12, Amsterdam, The Netherlands*, pages 532–539, 2005.

[13] Philip Welch. Characteristics of discrete transfinite time Turing machine models: Halting times, stabilization times, and normal form theorems. *Theoretical Computer Science*, 410:426–442, May 2009.

[14] Philip Welch. Characterisations of variant transfinite computational models: Infinite time turing, ordinal time turing, and blum–shub–smale machines. *Computability*, 10:1–22, 01 2021.