Infinite time busy beavers

Oscar Defrain, Bruno Durand, and Grégory Lafitte

LIRMM, CNRS, Université de Montpellier
161 rue Ada, 34095 Montpellier Cedex 5, France
{oscar.defrain, bruno.durand, gregory.lafitte}@lirmm.fr

Abstract. In 1962, Hungarian mathematician Tibor Radó introduced in [8] the busy beaver competition for Turing machines: in a class of machines, find one which halts after the greatest number of steps when started on the empty input. In this paper, we generalise the busy beaver competition to the infinite time Turing machines (ITTMs) introduced in [6] by Hamkins and Lewis in 2000. We introduce two busy beaver functions on ITTMs and show both theoretical and experimental results on these functions. We give in particular a comprehensive study, with champions for the busy beaver competition, of the classes of ITTMs with one or two states (in addition to the *halt* and *limit* states). The computation power of ITTMs is humongous and thus makes the experimental study of this generalisation of Radó's competition and functions a daunting challenge. We end this paper with a characterisation of the power of those machines when the use of the tape is restricted in various ways.

1 Introduction

Infinite time Turing machines (ITTMs) were defined by Hamkins and Kidder in 1989 and introduced in 2000 in a seminal paper [6] by Hamkins and Lewis. They are a generalisation of classical Turing machines to infinite ordinal time — the only difference is the behaviour at limit ordinal stages: the head moves back to the origin, the machine enters a special *limit* state, and each cell takes as value the *supremum limit* (abbreviated to *limsup*) of its previous values.

The busy beaver competition for Turing machines was presented by Hungarian mathematician Tibor Radó in 1962 [8]: in a class of machines, find one which halts after the greatest number of steps when started on the empty input. Radó also considered other complexity criteria, *e.g.*, the number of non-zero cells.

In this paper, we generalise the busy beaver competition to ITTMs and ordinal time. We introduce several busy beaver functions on ITTMs and show both theoretical and experimental results on these functions. The research in this paper is thus an original mixture of theoretical and experimental approaches,

The research for this paper has been done thanks to the support of the *Agence nationale de la recherche* through the RaCAF ANR-15-CE40-0016-01 grant.
The authors would also like to express their thanks to the anonymous referees, who made numerous suggestions and interesting remarks.

feeding each other. There are several subtle theoretical results helping the brute-force experimentation and the experimental results point in the direction of theoretical results which then need to be proved.

It is important to mention that the brute-force experimentation is very different from the kind of experimentation one comes across in the classical busy beaver case. In the classical case, the busy beaver functions that we try to "compute" are non-recursive. In the infinite-time case, our functions are not only non-recursive, they are not even arithmetic. We are able to go beyond this difficulty by inventing different heuristics and by having the program ask the user his analysis of the values of the cells for certain limit ordinal stages. In the classical case, if a machine halts, one just needs to run it for a long enough time to end up witnessing that it halts. In the infinite-time case, it is not obvious to simulate the machine for an unbounded ordinal number of steps, one needs to simulate the different kinds of ordinal limits to go higher up in the ordinal ladder. On the other hand, in the classical case, the number of steps reached are very large numbers without a clear meaning, but in the infinite-time case, the ordinals reached speak a lot more for themselves.

As in the classical case, one of the busy beaver complexity criteria for our infinite-time case is the number of ordinal steps. ITTMs have the particular property that every diverging machine eventually reaches a periodic behaviour, a *final loop* from which it will never escape. One of the other busy beaver complexity criteria is thus the stage at which a diverging machine enters this final loop. Another one is the *period* of that final loop. Each of these complexity criteria give rise to a distinct busy beaver function.

In section 3, we give a comprehensive study of these busy beaver functions for classes of ITTMs with one or two states (in addition to the *halt* and *limit* states). We analyse completely and in detail the behaviour of the machines of those two classes when started with an empty input and classify those machines in different categories, similar to the study made by Allen H. Brady for the first classes of the classical case [3] in the 1970's. The ITTM simulation itself is described at the beginning of section 3. The experimental simulation can be carried out and the different busy beaver functions are identified thanks to the theoretical results from section 2. By this experimental study, we obtain the champions for those two classes of these various busy beaver competitions.

The busy beaver functions are also studied from a degree-theoretic point of view in section 2 and we end this paper with characterisations of the power of those machines when the use of the tape is limited in various ways, characterisations hinted by the experimental study.

2 ITTMs and busy beaver functions

2.1 Infinite time Turing machines

An *infinite time Turing machine* (ITTM for short) is a three-tape (input, scratch, output) Turing machine on alphabet $\{0, 1\}$ with one *halt* state enhanced with a special *limit* state.

The machine halts when it has reached its *halt* state. The *limit* state makes it possible for a Turing machine to carry on for an infinite ordinal time. At a limit ordinal stage, an ITTM enters the special *limit* state, moves its head back to the first cell, and updates its tapes such that each cell is equal to the *limsup* of its previous values. After reaching this special *limit* state, it carries on its computation as a classic Turing machine before entering again the special *limit* state, and so on. We can thus let the machine run for an arbitrary large ordinal time.

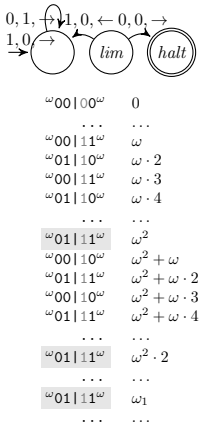

Fig. 1. ITTM loops

The *limsup* rule for the cells implies that if the value of a cell at a limit stage is 0, then there must be some earlier stage at which the cell had the value 0 and never subsequently changes. If the cell has the value 1 at a limit stage, then there are two possibilities, either at some earlier stage the cell obtained the value 1 and never subsequently changed, or the value of the cell alternated unboundedly often between the values 0 and 1 before that limit stage. To summarize, at a limit stage, a cell has either *stabilized* (to 0 or 1) or has value 1 because it *flashed*, *i.e.*, the cell alternated unboundedly often between values 0 and 1.

The power of these machines can be grasped through various ordinals. An ordinal α is *clockable* if there is an ITTM halting at this stage α starting with an empty tape.[1] An ordinal α is *writable* if there is an ITTM halting after having written a real coding the characteristic sequence of a well-order on ω of ordinal type α. The supremum of the clockable ordinals, denoted by γ_∞, and the supremum of the writable ordinals, denoted by λ_∞, can be shown to be equal and quite a large countable ordinal, well beyond ω_1^{CK} as it is the ${\lambda_\infty}^{\text{th}}$ admissible ordinal, the ${\lambda_\infty}^{\text{th}}$ admissible limit of admissible ordinals (recursively inaccessible), the ${\lambda_\infty}^{\text{th}}$ admissible limit of recursively inaccessibles, and so on.[2]

Two other ordinals which play an important role for these machines are ζ_∞ and Σ_∞ defined respectively as the supremum of eventually[3] writable and accidentally[3] writable ordinals. Philip Welch gave in [10] the characterisation for these ordinals: the triple λ_∞, ζ_∞, Σ_∞ is the lexicographically least triple α, β, γ such that $L_\alpha \prec_1 L_\beta \prec_2 L_\gamma$.

For classical Turing machines, when a machine enters a loop, it can never escape it. It is important to note that the behaviour of infinite-time Turing machines is completely different in that respect: a machine can enter a loop and finally escape it thanks to the *supremum limit* rule of the cell values. A loop

[1] The last transition is not counted to allow for clockable limit ordinals [6].

[2] Concerning admissible sets and ordinals, the interested reader is referred to [1].

[3] A real x is *eventually writable* if there is a non-halting infinite time computation, on input 0, which eventually writes x on the output tape and never changes again. A real x is *accidentally writable* if it appears on one of the tapes during a computation, possibly changing it again later on. An ordinal α is *eventually* (resp. *accidentally*) writable if the real coding a well-order on ω of order-type α is eventually (resp. accidentally) writable.

from which a machine cannot escape will be called a *final loop*. An example of a machine escaping a first loop and then being caught in an other loop, which is final, is given with parts of its space-time diagram[4] in Fig. 1. The following theorem states that every diverging machine eventually enters a final loop. Moreover, every ITTM either halts or repeats itself in countably many steps ([6, Theorem 1.1]). It also gives some explanations on how an ITTM can diverge. We sketch the proof of the first part because it gives fundamental insights in the workings of ITTMs and is essential to understand the rest of this paper.

Theorem 1 (Final Loop [6,10]). *Any halting ITTM halts at a countable stage. Moreover, any diverging ITTM eventually enters, at a countable stage, a final loop from which it will never escape. If the diverging machine runs on an empty tape, its configurations at time ω_1, Σ_∞ and ζ_∞ are equal. If the diverging machine runs on input x, its configurations at time ω_1, Σ^x_∞ and ζ^x_∞ are equal, where the two latter ordinals represent the relativised versions.*

The proof of the first part of this theorem can be found in [6]. It considers the configuration[5] C reached at stage ω_1. This configuration already appears at a countable stage γ and will be repeated unboundedly often. It is important to note that none of the cells which have stabilized to the value 0 at stage γ, and thus also at stage ω_1, will ever again turn to 1. So, the configuration at a limit stage before which the configuration C was repeated unboundedly often will again be the configuration C. The computation will thus repeat endlessly this *final loop* which started at stage γ and never halt. We consider the supremum of the enter-stages of these loops and using arguments from [10], we can show that it is in fact equal to ζ_∞. The supremum of the loop periods (and end-stages) of final loops can also be shown to be equal to Σ_∞.

This bound is reached by a variant of a universal machine: a machine $\mathcal{U}$ that simulates the computations of all ITTMs on the empty input, on ω many disjoint portions of the scratch tape. Moreover, $\mathcal{U}$ can be chosen to almost work in 'ω-real-time', *i.e.,* such that $\mathcal{U}$ simulates ω many steps of each machine in ω many steps of simulation time.

In the sequel, we say that an ordinal α is *looping* if there exists an ITTM that enters its final loop at stage exactly α. In other terms, the ω_1-configuration of this machine appears for the first time at stage α. A final loop example is given in Fig. 1, with a machine entering its final loop at stage ω^2, thus making this ordinal looping.

Now we can formulate some remarks on final loops that will be very useful for our experimental study. Let us consider the configuration of an ITTM when it is entering a final loop. Its 0's will remain at 0 forever while its 1's may change.

[4] ${}^{\omega}00|10^{\omega}$ denotes a tape with its left part filled with 0's and its right part filled with a 1 (the origin) and then 0's. In this notation, the origin is the first cell on the right of the symbol |, here a 1.

[5] The *configuration* of a computation at a given stage is a complete description of the *state* of the machine at this stage. It comprises the state, the position of the head and the complete content of the tapes.

Indeed, if a cell at 0 changes to 1 at least once during a final loop, consider this cell after ω loops. Its value is 1, which contradicts that the loop was chosen final. As a consequence, if two configurations at distinct limit stages α and β of an ITTM M are the same and if none of the 0's ever flash between those two stages, then M is diverging and the loop between α and β is final. This property will be used to decide divergence in our simulator.

2.2 Infinite time busy beaver functions

We define here an infinite time version of the well known *busy beaver* competition introduced by Tibor Radó in 1962 [8] for classical Turing machines.

The model considered in [6] has three tapes. For a more natural definition of busy beavers, we limit our model to one tape (with the *supremum limit* behaviour for cell values at limit stages). It is easy to see that restricting our model to one tape changes the clockable ordinal gaps, but that the supremums of clockable, eventually writable and accidentally writable ordinals remain the same.

Informally, the *infinite time busy beaver* competition consists, for a given n, in selecting the machine among the ITTMs with n states in addition to the *halt* and *limit* states (called the BB-*category* n of the competition) which *produces* the greatest ordinal when run from an empty input. Several busy beaver functions can be devised according to the different ways these ordinals can be produced.

We define three natural generalisations of the classical case, derived from the notions of clockable ordinals and final loops.[6]

We first define $\Gamma : \omega \to \gamma_\infty$, $\Lambda : \omega \to \zeta_\infty + 1$, and $\Pi : \omega \to \Sigma_\infty + 1$ as the functions that consider as input an ITTM and produce the ordinal stage at which it reaches respectively the *halt* state (if it halts), its final loop and the period of its final loop (if it diverges)[7].

We now define the BB-functions Γ_{BB}, Λ_{BB} and Π_{BB} as the functions which maximize the functions Γ, Λ, and Π on each BB-category. The first BB-function $\Gamma_{\text{BB}} : \omega \to \gamma_\infty$ maps a given BB-category n to the largest clockable ordinal *produced* by an ITTM of that category, *etc.*

In order to compare the *power* of these functions and of related ordinals, we use the ITTM-reduction[8], denoted by $\preccurlyeq_\infty$ and defined in [6], and assume that the ordinals considered (also as output of these functions) are coded in reals with an appropriate coding. For each ordinal α, there are continuum many possible choices for r_α, the real coding α. We would like to consider codes that do not contain extra information, in particular with regards to $\preccurlyeq_\infty$. r_α is thus chosen to be the L-least real coding a well-order on ω of order-type α. For the ordinals considered in this paper, this means that the code r_α considered for α will be such that $r_\alpha \in L_{\alpha+1}$.

[6] Loops can be considered as transient steps followed by an eventual final loop, and it is natural to measure the stage at which it appears and the length of that final loop.

[7] By Theorem 1, we know that the *sup* of looping ordinals, ζ_∞, and periods, Σ_∞, are reached by the universal machine. Λ and Π can thus reach these maximum values.

[8] A is ITTM computable from B, written $A \preccurlyeq_\infty B$, if the characteristic function of A is infinite time computable with oracle B. Reals are seen here as subsets of ω.

Remark that the functions Λ_{BB} and Π_{BB} (but not Γ_{BB}) will become eventually constant when the BB-category reaches a level containing a universal machine.

Theorem 2. $r_{\gamma_\infty} \equiv_\infty 0^\triangledown \equiv_\infty \Gamma_{\text{BB}} \equiv_\infty \Gamma$ *(using the notations[9] of [6])*
$\prec_\infty \Lambda_{\text{BB}} \equiv_\infty r_{\zeta_\infty} \equiv_\infty r_{\Sigma_\infty} \equiv_\infty \Lambda \equiv_\infty \Pi_{\text{BB}} \equiv_\infty \Pi \prec_\infty 0^\blacktriangledown$

Proof: Let's first prove that $r_{\gamma_\infty} \preccurlyeq_\infty 0^\triangledown$. The idea is to build an order such that $i < j$ if M_i halts before M_j. As ω machines halt at a given time, we consider only the first program with a given halting time that is discovered (while running the chosen universal ω-real-time machine $\mathcal{U}$). Our program runs as follows: it simulates $\mathcal{U}$ and little by little constructs an order such that $i < j$ if i and j are considered and if M_i halts before M_j. The representation of the order must be adjusted carefully : when we add an element in this order, then ω values are changed, but we can propagate those changes in parallel by runs of ω. In parallel we check whether all elements of $0^\triangledown$ have been observed halting in the $\mathcal{U}$-simulation. If yes, we halt (at time γ_∞). We thus get x that codes γ_∞, and by minimality we get $r_{\gamma_\infty} \preccurlyeq_\infty x \preccurlyeq_\infty 0^\triangledown$.

$0^\triangledown \preccurlyeq_\infty r_{\gamma_\infty}$: We consider e as input, simulate the computation of M_e on 0, and count-through r_{γ_∞}. Either we observe first that $M_e(x)$ halts and then $e \in 0^\triangledown$, or we observe that the count-through halts and then $e \notin 0^\triangledown$.

$0^\triangledown \preccurlyeq_\infty \Gamma$ is trivial since Γ contains $0^\triangledown$ and in addition, contains the halting time for all machines which halt.

$\Gamma \preccurlyeq_\infty 0^\triangledown$ corresponds to an important result of the literature [10]: any clockable ordinal is writable. In the proof of this theorem (see also [5] for an alternative construction) a single program transforms M_e that halts in α steps into a (minimal) code for α. We thus get the required inequality.

$\Gamma_{\text{BB}} \equiv_\infty \Gamma$: the proof is analogous to the Turing case. To ITTM-compute Γ_{BB} from Γ, one just needs to compute a maximum. In the other direction, one simulates all machines of the same class up to the time given by Γ_{BB} and as previously compute the *ad hoc* halting time if any.

We obviously have $r_{\gamma_\infty} \preccurlyeq_\infty r_{\zeta_\infty} \preccurlyeq_\infty r_{\Sigma_\infty}$. $r_{\gamma_\infty} \not\succcurlyeq_\infty r_{\zeta_\infty}$ since $\lambda_\infty^{r_{\lambda_\infty}}$ is eventually writable: we eventually compute r_{λ_∞} and in parallel, we simulate $\mathcal{U}$. When we observe that we are in a gap, we start computing $\mathcal{U}$ on the ordinal that starts the gap (obtained while eventually-computing r_{λ_∞}). With this latter simulation of $\mathcal{U}$, we build the order-type of the set of its clockables as usual. This process stabilizes when the input for $\mathcal{U}$ is r_{λ_∞} and eventually-computes $\lambda_\infty^{r_{\lambda_\infty}}$.

$r_{\Sigma_\infty} \preccurlyeq_\infty r_{\zeta_\infty}$ since once we have found the configuration at stage ζ_∞ of $\mathcal{U}$, we just need to wait for the second occurrence to grasp Σ_∞.[10] $\Lambda_{\text{BB}} \equiv_\infty r_{\zeta_\infty}$ is rather trivial since Λ_{BB} consists of a finite number of ordinals $< \zeta_\infty$ followed by r_{ζ_∞} itself. The first ones are just encoded as an integer in r_{ζ_∞} thus we add only a finite information that we can produce by a constant program. □

[9] We denote by M_e the e^{th} ITTM in a standard recursive enumeration. The two jumps are defined as $0^\triangledown = \{e \mid M_e(0)\downarrow\}$ and $0^\blacktriangledown = \{\langle e, x\rangle \mid M_e(x)\downarrow\}$.

[10] A corollary is that Σ_∞ is not admissible, as is already proven in [9, Corollary 3.4].

3 Experimental study of the busy beaver functions

3.1 Experimental simulation of an infinite time Turing machine

The experimental simulation of an ITTM is based on three parts. The first part is the simulation of the machine for one step. It simulates the read-write-move mechanism of the ITTM as any classical Turing machine simulator would do. The second part consists in guessing when to stop the simulation on successor ordinals, in order to jump to the next limit stage and devise the limit tape. It is based on the number of steps of computation as well as the machine's head trajectory (a great number of steps can be needed to reach such a threshold). The next limit stage itself is found by the simulator depending on current stages. The last part of the simulation is the guessing of the cell values of the limit tape considering the cells that have been flashed at previous stages. It is based on several criteria. A major one is cell stability, *i.e.,* analysing for each cell the number of times its value changed (till the last limit stage) and the number of computation steps that has elapsed till the last value change. Other criteria are the head trajectory for limits of successor ordinals, previous cofinal flashes[11] for higher limit ordinals, areas of differences between consecutive limit stages, *etc.* All these elements together allow us to understand the behaviour of the machine and to devise the limit tape. Finally, human intervention is possible and sometimes necessary if the simulator cannot make a decision on how to move up in the limit ordinal ladder or what values the cells of the limit tape should have.

As our simulation has obviously only finite means, there are many limits to our simulation. Considering previous flashing cells on the tape to build the cells' values at a limit stage, the simulator could recognise a stable repetitive pattern when in reality it would have changed a few thousand stages of computation later. This is especially the case when dealing with machines that present a chaotic behaviour (already present in the second category, BB-2). A solution is to increase memory thresholds (number of steps, tape size, *etc.*), at the expense of simulation speed. Finally, a lot of user double-checking by hand allowed us to build quite a reliable simulator (at least for machines with few states).

Clockable ordinals are simply obtained when reading the halting transition. Diverging machines and looping ordinals are detected using the divergence remark at the end of Section 2.1.

3.2 Study of whole classes of machines

The study of a whole class of machines mixes enumeration and simulation. We first explain our enumeration of machines, and give a summary of the various behaviours of these machines and largest clockable and looping ordinals found during the simulation of these classes.

[11] Finitely many flashes that occur at successor ordinals do not affect the next limit stage but later ones of higher order (*cf. cofinally flashing* behaviour in the sequel).

Enumerating a class of machines We enumerate the machines by adding transitions on the fly, making sure that we only add a transition when needed (in the simulation) and avoiding the different symmetries that can arise in the transitions graph. When an ITTM halts or is found as diverging, we simply update the transition that has been last set following a certain order on transitions. This order allow us to simulate a class completely.

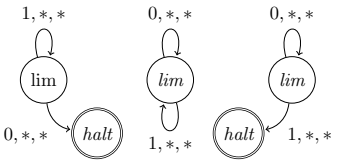

Fig. 2. ITTM specific filters

Moreover, filters are applied to get rid of non connected machines (in a sense of graph connectivity on the transitions graph), symmetric and trivial machines. This is very similar to what a classical Turing machine simulator would do. However, there is yet another filter specific to ITTMs which allows us to ignore patterns shown in Fig. 2 that lead either to a loop or to a halt at a stage that would not exceed[12] the limit ordinal $\omega \cdot 2$.

This enumeration on the fly allows us to find respectively 169 and 52189 distinct machines for the first two classes, BB-1 and BB-2.

Experimental results Experimental simulations gave us more material to appreciate how much ITTMs can be powerful even with only one or two states. Typical computational behaviours such as counters and fractal space-time diagrams already appear in the first and second categories, BB-1 and BB-2. Moreover, some behaviours of the classical busy beaver (such as *Xmas trees*, *cf.* [3]) appear with a smaller number of states in our ITTM competition, *e.g.*, Xmas trees with 1 state instead of 4.

We now give an overview of the first two classes with a selection of representative machines for each typical behaviour and their variants.

BB-1 This little class possesses 4 typical distinct behaviours, summarized in the following. The *Xmas tree* behaviour appears thanks to a previously written ${}^{\omega}0|1^{\omega}$ tape[4] after ω steps. The machine's head goes unboundedly often from left to right, drawing a Xmas tree (in its space-time diagram). The *cofinally flashing* behaviour is the ITTM's most natural behaviour: the machine flashes finitely many times a 0 cell and ends up stabilizing it to 0 (and thus without changing its value at the limit). Thus, if repeated on each limit, say ω, $\omega \cdot 2$, ..., these finitely many flashes directly affect the limit of limits, that is ω^2. The *noise expanding* behaviour consists in a machine generating noise at the origin, due to

[12] Indeed, suppose the machine did not halt for ω steps and reached the *limit* state. Either the machine will loop indefinitely on this *limit* state and is said to be diverging or will eventually halt excluding the middle pattern. Suppose it will eventually halt and go for more than ω steps taking the transition that loops on the *limit* state: either it will halt on the tape, reading a different symbol at a stage $\omega + c$ for an integer c, or reach the next *limit* stage at $\omega \cdot 2$ by reading a uniform tape. At this point, the machine halts if the looping transition modified the origin cell value, or loops indefinitely on the *limit* state.

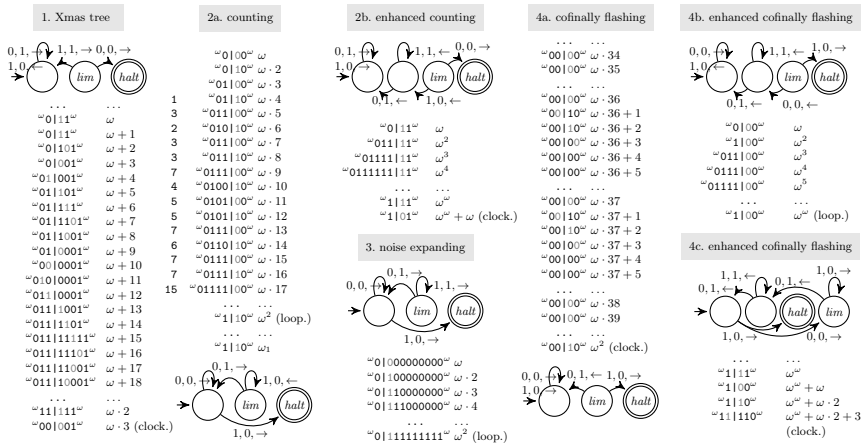

Fig. 3. Typical BB-1 behaviours (and their BB-2 extensions)

multiple limit jumps (and the head going directly to the origin). This noise grows and ends up taking all the tape at a higher limit. Finally, we denote by *counting* behaviour a machine that launches a counting procedure such as enumerating integers, in a certain order and written in binary, at limit tapes. These integers in binary can be viewed as patterns and their alternation affects limits of higher order.

The largest clockable ordinal for this class is produced by the *cofinally flashing* behaviour. The largest looping ordinals are produced by *cofinally flashing*, *noise expanding* and *counting* behaviours. For this first category BB-1, we get $\Gamma_{\text{BB}}(1) = \Lambda_{\text{BB}}(1) = \omega^2$ witnessed by the halting machine 4a and the diverging machines 2a ($\Pi(\cdot) = \omega^2$) and 3 ($\Pi(\cdot) = \omega$) in Fig. 3.

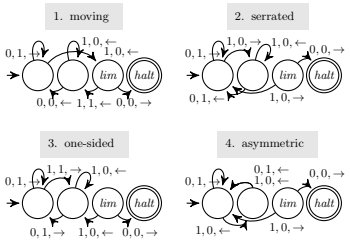

Fig. 4. Xmas BB-2 variants

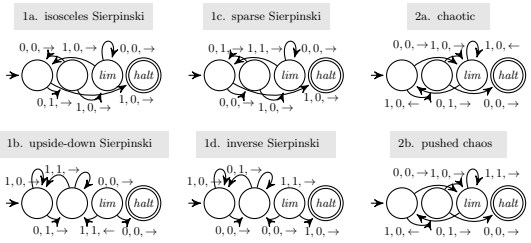

Fig. 5. New BB-2 behaviours

BB-2 Many enriched behaviours such as *moving* or *one-sided Xmas trees* are now possible, along with *fractal counting*, *etc.* Moreover, previous typical behaviours such as *cofinally flashing* and *counting* behaviours are now improved. Example machines for these enriched behaviours are given in Fig. 4 and 6. Improved aforementioned behaviours are given in Fig. 3, *e.g.*, the *'enhanced' counting* and *cofinally flashing* behaviours.

Two new typical distinct behaviours appear in this class. The first one draws Sierpinski triangles when considering a history of all flashes performed to reach $\omega \cdot c$ stages. We denote this phenomenon by the *Sierpinski* behaviour. The second

one is the *chaotic* behaviour that consists of a machine reducing or expanding concatenated groups of same values on each limit stage $\omega \cdot c$ for every integer c. This procedure generates a lot of flashes all over the tape. These flashes appear as being chaotic at a higher order limit stage and the machine promptly halts or diverges. Representative machines for these two behaviours are given in Fig. 5.

The largest clockable ordinal for this class is once again produced by the *cofinally flashing* behaviour with $\Gamma_{\text{BB}}(2) = \omega^\omega + \omega \cdot 2 + 3$ witnessed by the machine 4c in Fig. 3. The largest looping ordinal is produced by *cofinally flashing* and *counting* behaviours with $\Lambda_{\text{BB}}(2) = \omega^\omega$ witnessed by the machine 4b ($\Pi(\cdot) = \omega$) in Fig. 3.

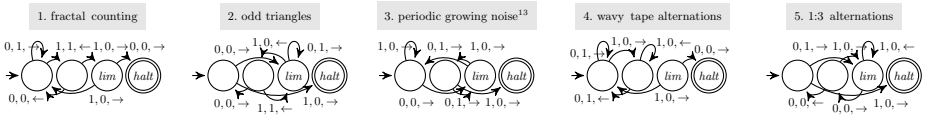

Fig. 6. Other BB-2 variants

4 Tape use restrictions

In this section, we investigate the consequences of several restrictions on the kind of tape that the machines are allowed to use.

4.1 Finite tape restriction

We say that an ITTM M *uses* k *cells on input* x, if M halts on the entry x, and if there exists a set of exactly k cells such that outside this set, the cells remain at value 0 during all the computation.

Proposition (Only one cell). The possible halting times for machines using one cell on input 0 are exactly n, $\omega + n$, $\omega \cdot 2 + n$, or $\omega^2 + n$ where n is any finite constant.

Proof: The halting times $\omega + n$ and $\omega \cdot 2 + n$ are straightforward (to get the $+n$ term, we use n extra states). In Fig. 3 we present the winner of the class BB-1 that halts after exactly ω^2 steps (and then add $+n$). The halting times $\omega \cdot k$, $k \geq 3$, and every $\alpha > \omega^2 + \omega$ are impossible since there only two possible choices for the value of the origin cell (when in the *limit* state). □

Theorem 3 (Finitely many cells). *The ordinals clockable by machines using finitely many cells are exactly the ordinals* $< \omega^\omega$.

Proof: We assume that there exists an ITTM M that halts in ω^ω or more steps using k cells and obtain a contradiction. Consider the configuration at stage ω^ω and consider a window of finite size that contains all k cells touched by M.

There exists a stage γ after which all 0's in the window (identified as the *grey* cells) placed on the configuration at stage ω^ω have stabilized. Between γ and ω^ω,

[13] This halting machine ends up displaying a periodic tape ${}^\omega 0|(1110)^\omega$ at stage ω^2.

all limit configurations have the grey cells at 0. Among these limit configurations, let us consider pairs of equal configurations. We observe their number of 1's. It has a *supremum limit* denoted by S. If S is exactly the number of 1's of the configuration at stage ω^ω, then there exists a pair of configurations where all 0's from inside the window are grey cells. From the remarks at the end of Section 2.1, we conclude that M diverges. Thus S is less than the number of 1's of the configuration at stage ω^ω. Let us denote by β_1 and β_2 the stages corresponding to such a pair with S 1's. If all 0's of the window remain unchanged between stage β_1 and β_2, then again M diverges. Thus at least one non-grey 0 flashes at least once. Now let us consider ω times this (non-final) loop between β_1 and β_2. As β_2 is bounded by ω^a for some integer a, we get a limit configuration with at least $S+1$ 1's before ω^ω. After this configuration, we consider another pair β_1' and β_2' with S 1's. By the same argument we get another limit configuration with at least $S+1$ 1's before ω^ω. We iterate this process, and, having only a finite choice for these 1's in the window, we get two equal limit configurations with $S+1$ 1's which contradicts the definition of S.

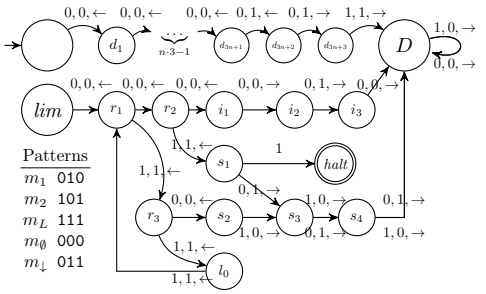

Fig. 7. Machine halting after ω^n steps

We now construct a machine that halts after time ω^n for any integer n using only a finite number of cells. This machine is based on the improved *counting* behaviour described in Section 3.2, and is intentionally left unoptimized for a better understanding. In the following, we denote by the term 'pattern' a set of 3 consecutive cells described in the tabular in Fig. 7. In the sequel, we explain how this machine uses them to iterate a counter up to ω^n.

First of all, a delimiter pattern $m_\downarrow$ is written $3n$ cells away at left from origin and the machine enters the D-state (*cf.* Fig. 7) that goes quietly to the ω limit without flashing any new cells. This is denoted by states $(d_1, \ldots, d_{3n+3})$ in Fig. 7. Then, a counting procedure is launched, based on pattern recognition and alternation. If the machine recognises an empty pattern $m_\emptyset$ (r_1, r_2, r_3 states), it remplaces it with a m_1 pattern (i_1, i_2, i_3) and jumps to the next limit. If one of the standard patterns (m_1 and m_2) are recognised, the pattern is switched from m_1 to m_2 or reciprocally (s_1, s_2, s_3) and the machine loops to the next limit. Although it is never written, a limit pattern m_L will be found after ω alternations of m_1 and m_2 and thus m_L appears for the first stage after ω^2 steps of computation. If m_L is read, then the machine ignores it and seeks the next pattern on the left (l_0). Using one state (D-state), we erase all m_L patterns after next patterns incrementation. When a limit pattern m_L is found for the first time in the k^{th} pattern from the origin, the machine has computed ω^k steps. The whole counting procedure stops when the tape is full of m_L patterns between origin and $m_\downarrow$, which is discovered at stage $\omega^n + 3n + 2$ using $3n + 4$ cells. □

4.2 Ultimately periodic and recursive tape restrictions

We focus on those machines whose tapes are ultimately periodic at any computation time. We can define those machines as those for which their configurations at any limit stage is ultimately periodic.[14] One reason for considering these machines is that at any time, their tape can be finitely described.

Of course, the natural extension of these two tape-restriction notions is to ask for the tape at any stage to be recursive. This is also a constraint that only concerns limits stages: a point-to-point limit of those recursive tapes must be recursive.[15]

We conjecture the following generalisation of Theorem 3.

Conjecture. *Every ITTM which halts when started on the empty input and whose tapes are ultimately periodic (resp. recursive) halts after less than ε_0 (resp. ω_1^{CK}) steps.*

References

1. Barwise, J.: Admissible Sets and Structures: An Approach to Definability Theory, Perspectives in Mathematical Logic, vol. 7. Springer-Verlag (1975)
2. Brady, A.H.: The conjectured highest scoring machines for Radó's $\Sigma(k)$ for the value $k = 4$. IEEE Transactions on Electronic Computers EC-15, 802–803 (1966)
3. Brady, A.H.: The determination of the value of Radó's noncomputable function $\Sigma(k)$ for four-state Turing machines. Mathematics of Computation 40(162), 647–665 (April 1983)
4. Brady, A.H.: The busy beaver game and the meaning of life. In: Herken, R. (ed.) The Universal Turing Machine: A Half-Century Survey, pp. 237–254. Springer-Verlag, Wien, New York, second edn. (1995)
5. Durand, B., Lafitte, G.: A constructive swissknife for infinite time Turing machines (2016)
6. Hamkins, J.D., Lewis, A.: Infinite time Turing machines. Journal of Symbolic Logic 65(2), 567–604 (2000)
7. Lafitte, G., Papazian, C.: The fabric of small Turing machines. In: Computation and Logic in the Real World, Proceedings of the Third Conference on Computability in Europe, CiE 2007, June 18-23, Siena, Italy. pp. 219–227 (2007)
8. Radó, T.: On non-computable functions. Bell System Technical Journal 41(3), 877–884 (May 1962)
9. Welch, P.D.: The length of infinite time Turing machine computations. Bulletin of the London Mathematical Society 32, 129–136 (2000)
10. Welch, P.D.: 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] Note that knowing whether a machine has this property is not decidable. This was also the case with the finite tape restriction.

[15] The situation is not the same for stage ω than for compound limit ordinal stages, *e.g.,* ω^2, since in the first case we take the limit of a recursive sequence of recursive tapes while in the latter, the sequence of tapes is not necessarily recursive.