Archive for Mathematical Logic (2020) 59:335–365
https://doi.org/10.1007/s00153-019-00692-9

Weaker variants of infinite time Turing machines

Matteo Bianchetti[1]

Received: 3 June 2017 / Accepted: 6 July 2019 / Published online: 13 September 2019

Abstract

Infinite time Turing machines represent a model of computability that extends the operations of Turing machines to transfinite ordinal time by defining the content of each cell at limit steps to be the lim sup of the sequences of previous contents of that cell. In this paper, we study a computational model obtained by replacing the lim sup rule with an 'eventually constant' rule: at each limit step, the value of each cell is defined if and only if the content of that cell has stabilized before that limit step and is then equal to this constant value. We call these machines weak infinite time Turing machines (wITTMs). We study different variants of wITTMs adding multiple tapes, heads, or bidimensional tapes. We show that some of these models are equivalent to each other concerning their computational strength. We show that wITTMs decide exactly the arithmetic relations on natural numbers.

Keywords Ordinal computability · Infinite time Turing machine · Transfinite computation · Supertask · Arithmetic hierarchy · Real arithmetic

Mathematics Subject Classification 03D10 · 03D60 · 03D78 · 68Q01

Contents

✉ Matteo Bianchetti
matteobianchetti@gmail.com

1 Department of Philosophy, University of Notre Dame, 100 Malloy Hall, Notre Dame, IN 46556, USA

1 Introduction

Hamkins and Lewis suggested a new theoretical computing device that they called *infinite time Turing machine* (for short: ITTM). They first described machines of this type in [5]. One can roughly describe ITTMs as Turing machines that can halt and provide a result after infinitely many steps. ITTMs are far more powerful than ordinary Turing machines. The basic operations and relations over the reals are ITTM-computable, and the halting set is ITTM-decidable. In fact, every Π^1_1 set is ITTM-decidable. One attractive feature of ITTMs is their resemblance to ordinary Turing machines, which allows for "implementational" (although often lengthy) proofs of statements describing what ITTMs can and cannot do, i.e. proofs explaining the behavior of a machine of this type attempting to carry out a specific computation.

In this paper, we investigate models of infinitary computation whose computational power lies strictly between that of Turing machines and ITTMs. In particular, we define four types of weak infinite time Turing machines, showing results about their computational power. We prove that these machines can carry out basic operations and decide basic relations over the reals, and their halting time is bounded above by ω^2. We also prove some results on the computational strength of these wITTMs and their variants. Although one could provide shorter proofs, we will often exploit the "mechanical" character of these models of computation and prove statements by describing programs for these machines to carry out the relevant operations.

2 Weak infinite time Turing machines

After briefly recalling the definition of ITTM, in this section we define a variant of machines of this type. We call a machine of this new type *weak infinite time Turing machine* (wITTM). We prove some basic results about the halting time and the computational power of wITTMs. Among other things, we prove that wITTMs are computationally strictly more powerful than ordinary Turing machines, can carry out basic operations and decide basic relations over the reals, and are computationally strictly weaker than ITTMs.

2.1 Infinite time Turing machines

An ITTM consists of three tapes, each having a leftmost cell and being infinitely extended to the right. The three tapes are arranged as in Fig. 1. From top to bottom, we will refer to these tapes, respectively, as the *input* tape, the *scratch* tape, and the *output* tape. Each machine has a read–write head that, at each step of the computation, reads the content of a vertical slice consisting of three cells (one on the input tape, the one below it on the scratch tape, and the one below that one on the output tape), consults the program associated with the machine, and carries out the relevant action (if any). The tape symbols are 0 and 1. As in an ordinary Turing machine, the possible actions are: printing a new tape symbol in a cell, moving left one cell, and moving right one cell. At the beginning of the computation, the input (a sequence of 0s and 1s from 2^ω)

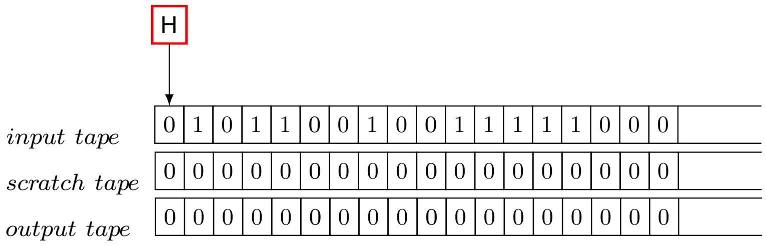

Fig. 1 Infinite time Turing machine

is on the input tape, while the scratch tape and the output tape contain 0 everywhere, as in Fig. 1. The output of a computation is the sequence of 0s and 1s on the output tape when the machine halts (if at all).

The first notable difference between an ITTM and an ordinary Turing machine is that the computation of an ITTM can converge after infinitely many steps. In other words, an ITTM can halt and provide an answer at step α, where α can be an infinite ordinal. At *successor steps*, i.e. at steps like 5 and $\omega + 2$, an ITTM behaves like a Turing machine. However, to ensure that the configuration of the machine is defined at *limit steps*, i.e. at steps such as ω and ω^2, we adopt the following rules:

(*Head*) At every limit step α the head is placed on the leftmost slice of cells, and it enters a unique special state called the *limit state*.

(*Lim Sup*) For every limit step α, for every cell c, the content of c at α is the lim sup of the values that have previously appeared in c.

According to *Head*, the machine is always in the same state at a limit step. At that point, the machine will read the content of the leftmost vertical slice of cells and consult the program for its next move. According to *Lim Sup*, at any limit step α, the value of any cell c at α is always defined and univocally determined by the previous values that have appeared in c. If the value in c stabilizes before α, then the value in c at α is that constant value. If the value in c keeps on changing between 0 and 1 before α, then the value in c at α is 1. Writing $c[x]$ for the content of c at step x, we can then write succinctly: for every limit step α,

$$c[\alpha] = \begin{cases} 1 & \text{if } \forall\beta < \alpha \ \exists\gamma \in (\beta, \alpha) \ \ c[\gamma] = 1 \\ 0 & \text{otherwise.} \end{cases}$$

2.2 Definition of weak infinite time Turing machines

The rule *Head* (or something essentially equivalent to it) is necessary for machines with the same hardware and programs of ITTMs to compute for infinitely many steps and possibly eventually halt. However, one could meaningfully replace the rule *Lim Sup* by some different stipulation. The following rule is a natural replacement of *Lim Sup*:

(*Eventually Constant*) For every limit step α, for every cell c, the content of c at α is defined if and only if it has stabilized before α. In this case, the content of c at α is that eventually constant value.

More formally, for every limit step α,

$$c[\alpha] = \begin{cases} 1 & \text{if } \exists \beta < \alpha \ \forall \gamma \in (\beta, \alpha) \ \ c[\gamma] = 1 \\ 0 & \text{if } \exists \beta < \alpha \ \forall \gamma \in (\beta, \alpha) \ \ c[\gamma] = 0 \\ \uparrow & \text{otherwise.} \end{cases} \tag{1}$$

If the content in some cell c does not stabilize before α, then the configuration of the machine at α is undefined, and the computation fails in the sense that it does not converge to any output. In this case, if the computation is defined for every $\beta < \alpha$, we say that the machine *hangs* at α (although, strictly speaking, the machine never reaches step α).[1]

It is useful to remark the following. Just as in the case of regular Turing machines, one can replace the alphabet $\{0, 1\}$ with any finite alphabet without modifying the computational power of wITTMs and without changing the computational power of any variant of wITTMs that we will consider later. One can code symbols using strings of 0s and 1s and the rule *Eventually Constant* works as usual. We will use this fact to simplify some proofs.

2.3 Halting time

It is useful to determine the halting time of wITTMs to discuss their computational power. The following results show that wITTMs can halt at any time before step ω^2 but not at step ω^2 or after that.

Lemma 1 *Let M be a wITTM and α either a limit step or the initial step. If the configuration of M is defined at step α and M does not change the content of any cell between step α and $\alpha + \omega$, then M will compute forever.*

Proof Suppose that the configuration of M is defined at step α and that M does not change the content of any cell between step α and step $\alpha + \omega$. Therefore, the configuration of M at step $\alpha + \omega$ is the same as the configuration of M at step α. Thus, between step $\alpha + \omega$ and step $\alpha + \omega \cdot 2$, M will behave as it did between step α and step $\alpha + \omega$, i.e. it will change the content of no cell. From this, it follows that the configuration of M at any limit step after α is the same as the configuration of M at α. Therefore, after each limit step, M will behave identically, i.e. it will change the content of no cell before the next limit step. It follows that M will never change the content of any cell after α. Since the configuration of M is defined at α, the configuration of M

[1] After I gave a talk on wITTMs at the Computability in Europe 2015 meeting, Merlin Carl pointed out to me that he and Joel D. Hamkins briefly discussed machines of this type in 2012 on MathOverflow (see https://mathoverflow.net/questions/111902/transfinitely-iterated-limit-computability). In that discussion, Hamkins stated the results that we here present as Theorems 2 and 26 and hinted briefly at their proofs. Apparently, Hamkins did not publish anything on wITTMs. Carl discusses wITTMs in chapter 2 of his forthcoming [3]. There Carl also presents a complete proof of Theorem 2, which he communicated to me.

is also defined at every limit step after α. Therefore, the machine will never stop and will compute forever. □

Theorem 2 (Hamkins) *Let M be a wITTM. M cannot change the content of any of its cells after step ω^2.*

Proof By Lemma 1, if M does not change the content of any cell between two successive limit steps, then M will loop forever without changing the content of any cell. So, it remains to consider whether M can change the content of a cell finitely many steps after ω^2. Suppose, toward a contradiction, that, for some $k \in \omega$, the content of cell c changes at step $\omega^2 + k$. Therefore, the configuration of the machine was defined at step ω^2. In particular, the content of no cell has changed unboundedly often before step ω^2. Let S be the finite set of the cell that the head inspects between step ω^2 and step k. Since the configuration of the machine is defined at step ω^2, the content of each cell in S has stabilized before ω^2. Since S is finite, there is a least ordinal $\alpha < \omega^2$ after which the content of no cell in S changes before ω^2. Therefore, at step ω^2, the relevant initial portions of the tapes of M are identical to the same portions of these tapes at every limit step β between α and ω^2. Since, at every limit step, M is in the limit state, for every limit step $\beta \in (\alpha, \omega^2)$, the content of cell c changed at step $\beta + k$. Therefore, the content of cell c has changed unboundedly often before step ω^2. However, this means that the configuration of the machine is not defined at step ω^2. This is a contradiction. Therefore, no cell can change after step ω^2. □

From Theorem 2, it follows that the halting time for wITTMs is less than ω^2.

Theorem 3 *For every wITTM M, for every input $f \in 2^\omega$, if M halts on input f, then it halts before step ω^2.*

One can further observe that a wITTM can hang exactly at step ω^2.

Proposition 4 *There is a wITTM M whose configuration is defined for every step $\alpha < \omega^2$ and that hangs at step ω^2.*

Proof Let M be a wITTM that, on any input, after step 0 and after each limit step $\beta < \omega^2$, prints 1 and then 0 on the first cell c of the scratch tape. After that action, M moves right without changing anything before the next limit step. At every step $\alpha < \omega^2$, the content of each cell has changed only finitely often. However, the content of c changes unboundedly often before step ω^2. Therefore, the machine hangs at step ω^2. □

From Theorem 2 and Proposition 4, one obtains the following result.

Halting Time Theorem 5 *ω^2 is the set of all halting times of wITTMs.*

A result analogous to Proposition 4 holds for loops in a computation.

Proposition 6 *There is a wITTM M that begins looping exactly at step ω^2.*

Proof Let M be a wITTM that, on any input, after step 0 and after each limit step α, looks for the leftmost 0 on the scratch tape and replaces it with 1. After that action, M moves right without changing the content of any cell before the next limit step. For every limit steps $\alpha < \omega^2$, the configuration of M at α is not the same as its configuration at $\alpha + \omega$. However, for every limit step $\beta \geq \omega^2$, M is always in the same configuration and simply scans the scratch tape and moves right. $\square$

2.4 Computational power of weak infinite time Turing machines

One can easily observe that wITTMs are more powerful than ordinary Turing machines.

Proposition 7 *There is a wITTM that computes the halting set K.*

Proof On input e, such wITTM tries to compute $\varphi_e(e)$. If the computation converges before step ω, then the machine writes 1 on the leftmost cell of the output tape and halts. If the computation does not converge before step ω, the machine enters the limit state at step ω and then halts at step $\omega + 1$. $\square$

In general, wITTMs can compute, in exactly $\omega + 1$ steps, every function that is not Turing computable but is limit computable. Therefore, for example, identifying reals with sequences in 2^ω, one can observe that wITTMs can carry out basic operations over the reals (such as addition and multiplication) and decide basic relations on the reals (such as identity and order).

One can also easily observe that ITTMs can compute everything that wITTMs compute. We will see later that the opposite is not true and, therefore, the computational power of wITTMs lies strictly between that of ordinary Turing machines and that of ITTMs.

2.4.1 Real arithmetic

Though it is easy to show that basic operations and relations over the reals are wITTM-computable, it is interesting to consider this fact more in-depth. Representing reals as infinite strings of 0s and 1s, real arithmetic is not Turing computable. However, there are algorithms to carry out addition and multiplication and to decide the order relation on reals, if we represent reals using segments. For example, at the beginning of *La geometrie*, René Descartes provides an algorithm for multiplying two positive reals r_0 and r_1 represented as segments BC and BD, when $r_1 > 1$ and a further unitary segment AB is given, as in Fig. 2.[2] The length of BC is r_0 and the length of BD is r_1. First one joins A with C. Then one draws a parallel to AC trough D. In this way, one determines the segment BE whose length is $r_0 \cdot r_1$.

One can carry out such a construction using just an unmarked ruler and a collapsible compass, as Euclid showed in *Elements* I 1–3. One can find completely general algorithms (but relying on a rigid compass and in the context of intuitionistic mathematics) in §8 of [1]. Let us now go back from geometric representations of reals to

[2] See [4] p. 298.

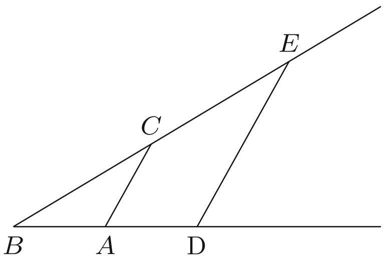

Fig. 2 Descartes's method for multiplying segments

their representations as binary sequences. It is instructive to examine which algorithm one could give to a wITTM to compute basic operations and relations over the reals. In this section, we will briefly describe how a wITTM could behave to carry out addition and multiplication over the reals.

For the sake of simplicity, we concentrate on reals in the interval [0, 1). Given $f \in 2^\omega$, we take it to represent the real

$$\sum_{i=0}^{\infty} f(i)2^{-i-1}.$$

Since different sequences could represent the same real (like $1\bar{0}$ and $0\bar{1}$), we single out a preferred representation, which we call *normal form*. A sequence $f \in 2^\omega$ is in normal form if and only if it does not end with a tail of 1s, i.e., for every n, for some $m > n$, $f(m) = 0$. The following results hold.

Lemma 8 *For some wITTM M, for every sequence f and $g \in 2^\omega$, M decides whether f is in normal form.*

Proof (Sketch.) A wITTM M starts by reading the input tape. Every time M finds a cell i on the input tape that contains 0, M looks for the leftmost cell on the scratch tape that contains 0 and prints 1 in that cell. Then M resumes examining the input tape from cell $i + 1$. At step ω, M checks whether the scratch tape contains any 0. If yes, M halts saying that f is not in normal form. If M reaches step $\omega 2$, it halts at step $\omega 2 + 1$ saying that f is in normal form. □

Lemma 9 *For some wITTM M, for every sequence $f \in 2^\omega$ that is not in normal form, M replaces f with a sequence $f' \in 2^\omega$ such that f' is in normal form and f and f' represent the same real.*

Proof (Sketch.) A wITTM M starts by reading the input tape. Every time M finds a cell i on the input tape such that i contains 0 and $i + 1$ contains 1, M prints 1 on cell number i on the scratch tape and prints 0 everywhere on the scratch tape to the left of i. Then M resumes reading the input tape from cell $i + 2$. At the limit step, M starts copying the sequence given on the input tape to the output tape and also checks the scratch tape. When M finds a cell c on the scratch tape that contains 1, M prints 1 on the output tape right below c and halts. □

Lemmas 8 and 9 show that a wITTM can decide whether a sequence of 0s and 1s is in normal form and, if not, it can replace it with an equivalent sequence that is in normal form. Therefore, we will always assume that all sequences $f \in 2^{\omega}$ that we consider are in normal form.

Let us write $r(f)$ for the real number represented by $f \in 2^{\omega}$. Equality and order are wITTM-computable, as the following result states.

Proposition 10 *For some wITTM M, for every sequence f and $g \in 2^{\omega}$, M decides which of the following relations holds: $r(f) = r(g)$, $r(f) < r(g)$, or $r(g) < r(f)$.*

Moreover, addition over the reals is wITTM-computable. To avoid uninformative complications, we describe a procedure to compute addition when the addends are between 0 and $\frac{1}{2}$.

Proposition 11 *For some wITTM M, for every sequence f and $g \in 2^{\omega}$ such that $r(f)$ and $r(g) \in [0, \frac{1}{2})$, M halts on output h such that $r(h) = r(f) + r(g)$.*

Let us write $f + g$ to refer to such h. For simplicity, we assume that $r(f)$ and $r(g) \in [0, \frac{1}{2})$. However, it will be clear, that a similar idea works for the general case as well. The idea for the proof is to compute $f + g$ as the limit of the partial sums $f_{|n} + g_{|n}$, where $f_{|n} = f(0)f(1)\dots f(n)$ and similarly for $g_{|n}$. To consider f and g at the same time one could code them in a unique binary sequence $f(0)g(0)f(1)g(1)\dots f(n)g(n)\dots$.[3] Notice that, while for every n, an ordinary Turing machine can compute $f_{|n} + g_{|n}$, $f + g$ is not Turing computable. Moreover, for some f and g, $(f+g)(n)$ is not Turing computable. For example, let $f = 010101\dots$ and $g = 0010101\dots$. At no step of its computation can a Turing machine decide whether $(f + g)(1) = 0$.

In working with wITTMs, it is necessary to exercise caution so that the program developed does not lead the machine to hang on a specific input. In the case of addition, the machine will change the content of some cells on the output tape to take care of carries. In other words, when the machine computes $f_{|n} + g_{|n}$, it will write the result on the output tape. Then it computes $f_{|n+1} + g_{|n+1}$ and writes the result on the output tape, sometimes changing the content of cells on the output tape that it used before. The following lemma addresses this point and is essential to complete the proof of Proposition 11.

Lemma 12 *For M as in Proposition 11, for every sequence f and $g \in 2^{\omega}$ such that $r(f)$ and $r(g) \in [0, \frac{1}{2})$, the content of every cell of M stabilizes before step ω.*

Proof Let us suppose that M has written $f_{|n} + g_{|n}$ on the output tape. To compute $f_{|n+1} + g_{|n+1}$, M first computes $f(n+1) + g(n+1)$. If either $f(n+1) = 0$ or $g(n+1) = 0$, then there are no carries to consider. In this case, M simply writes $f(n+1)+g(n+1)$ on cell number $n+1$ of the output tape. If $f(n+1) = g(n+1) = 1$, then M goes left until it finds 0 on the output tape. Let us say that it first finds 0 at cell number m, for some $0 \leq m \leq n$. Therefore, M writes 1 in cell number m on the

[3] We will later show that one could also imagine to have two input tapes at disposal. Adding finitely many input tapes does not alter the computational power of wITTMs. See, in particular, Equivalence Theorem 15.

output tape and writes 0 in all the cells strictly between cell m and cell $n+1$ on the output tape. One observes that a cell like m containing 0 and strictly to the left of cell number $n+1$ on the output tape always exists: since $r(f)$ and $r(g) \in [0, \frac{1}{2})$,

$$f_{|n} + g_{|n} < \sum_{i=1}^{n} 2^{-i},$$

where the cells used to write down $f_{|n} + g_{|n}$ are exactly those on the output tape between cell 0 and cell n (cell 0 and cell n included). More explicitly, one observes, first, that $f_{|0} + g_{|0} = 0$. Assume then that the required m exists for the computation up to $f_n + g_{|n}$. If M performs a carry computing $f_{|n+1} + g_{|n+1}$, then cell $n+1$ on the output tape will contain 0. If M does not perform a carry computing $f_{|n+1} + g_{|n+1}$, then the same cell m from before still contains 0. One observes that the rewriting required for taking care of carries stops at the rightmost cell m on the output tape that is to the left of cell $n+1$ and that contains 0. In particular, the value in each cell can change at most three times: first to write 1, then to write 0 taking care of a carry, and finally to write 1 again taking care of another carry. □

Now we briefly consider the case of multiplication, which is also wITTM-computable.

Proposition 13 *For some wITTM M, for every sequence f and $g \in 2^\omega$ such that $r(f)$ and $r(g) \in [0, 1)$, M halts on output h such that $r(h) = r(f) \cdot r(g)$.*

Proof Let us write fg to refer to such h. For simplicity, we assume that $r(f)$ and $r(g) \in [0, 1)$. However, an idea similar to the one that we are going to present works in the general case. The idea for the proof is to compute fg as the limit of the partial products $f_{|n}g_{|n}$. In particular, we can re-use the algorithm for addition to compute multiplication, since we have

$$fg = \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \frac{f(i)g(j)}{2^{i+j+2}}.$$

In particular, assuming that we have already computed $f_{|n}g_{|n}$, we compute $f_{|n+1}g_{|n+1}$ as

$$f_{|n}g_{|n} + \sum_{i=0}^{n} \frac{f(i)g(n+1)}{2^{i+n+3}} + \sum_{j=0}^{n+1} \frac{f(n+1)g(j)}{2^{n+j+3}}.$$

It remains to show that the content of no cell changes unboundedly often while we compute these partial sums. We can reason as in Lemma 12, provided that we show that, on tape t, where we record the output of the relevant sum, there is a cell c to the left of the rightmost cell on t used to record that sum such that c contains 0. Since, for every n,

$$f_{|n} g_{|n} < \sum_{i=1}^{2n+2} 2^{-i},$$

there is a cell c with the desired property. □

3 Variants of wITTMs

Weak infinite time Turing machines have exactly three tapes (input tape, scratch tape, and output tape) and one read–write head. Describing the algorithms for carrying out real arithmetic, we have hinted at the fact that one could add a finite number of extra input tapes and scratch tapes to these machines without altering their computational power. In this section, we clarify what this means and consider further modifications of the original hardware of wITTMs. In particular, we consider machines that work as wITTMs but the hardware of which are the following:

1. Weak infinite time Turing machines with finitely many extra input tapes and scratch tapes and a single head (for short, wITTM$_{<\omega,<\omega}$)
2. Weak infinite time Turing machines with finitely many extra pairs of input tapes and scratch tapes and a different read–write head for each pair (for short, wITTM$_{<\omega}$)
3. Weak infinite time Turing machines with a bidimensional input tape and a bidimensional scratch tape (for short, wITTM$_{\omega}$).

3.1 Finitely many tapes and single head

A wITTM$_{<\omega,<\omega}$ looks like the machine represented in Fig. 3, where $i_0, \ldots, i_n$ are the input tapes, $s_0, \ldots, s_m$ are the scratch tapes, and o the output tape. At successor steps, a wITTM$_{<\omega,<\omega}$ reads a vertical slice of the tape, consults the program, and carries out the required action (if any). At limit steps, a wITTM$_{<\omega,<\omega}$ behaves like a wITTM. Every wITTM is also a wITTM$_{<\omega,<\omega}$. Therefore, every function that is wITTM-p.c. is also wITTM$_{<\omega,<\omega}$-p.c.. The following theorem states that the reverse is also true, i.e. every function that is wITTM$_{<\omega,<\omega}$-p.c. is also wITTM-p.c. Let us write $M(r) = s$ to mean that the machine M on input r halts with output s. Let us also write $M(N(r))$ to mean that the machine M runs on the output of the machine N on input r.

Theorem 14 *For every $n \in \omega$, for some wITTM$_{<\omega,<\omega}$ N' with exactly $n+1$ input tapes, for every wITTM$_{<\omega,<\omega}$ N with exactly $n+1$ input tapes, for some wITTM M, for every $r \in (2^\omega)^{n+1}$, $N'(r) \in 2^\omega$ and*

1. *if $N(r)\downarrow$, then $M\big(N'(r)\big) = N(r)$ and*
2. *if $N(r)\uparrow$, then $M\big(N'(r)\big)\uparrow$.*

Proof Given $r = (r_0, \ldots, r_n)$, we define

$$N'(r) = r_0(0)r_1(0)\ldots r_n(0)r_0(1)r_1(1)\ldots r_n(1)\ldots r_0(m)r_1(m)\ldots r_n(m)\ldots.$$

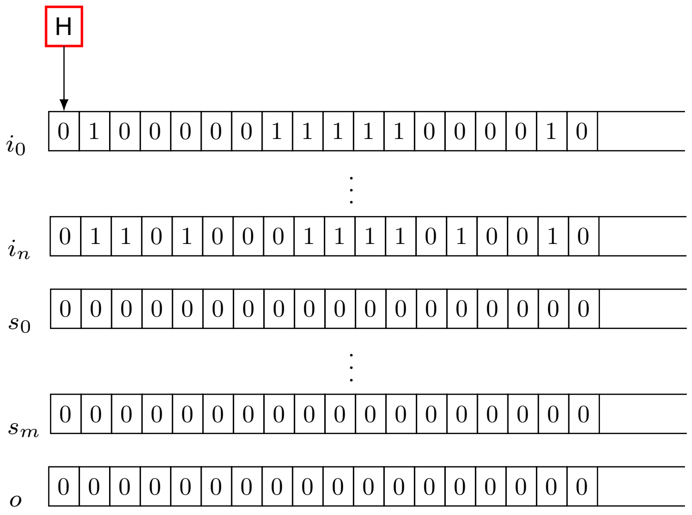

Fig. 3 Weak infinite time Turing machine with finitely many extra input tapes and finitely many extra scratch tapes

More formally, we define $N'(r)(x) = r_i(j)$ such that $x = j(n+1) + i$ for some $0 \leq i \leq n$ and some $j \in \omega$. By the Euclidean algorithm, such i and j exist and are unique.

We now describe how M on input $N'(r)$ mimics the behavior of N on input r. Let us write i_k to refer to the input tape number k of N and I to refer to the input tape of M. Similarly, we write s_k for the scratch tape number k of N and S for the scratch tape of M. We write o for the output tape of N and O for the output tape of M. For the sake of simplicity, let us assume that N has exactly $n+1$-many input tapes and $m+1$-many scratch tapes. Without loss of generality, we can further assume that $n = m$. At limit steps, both N and M behave in the same way. It remains to define the behavior of M at successor steps and to ensure that, when M halts, the output tape of M is identical to the output tape of N when N halts. We take care of these two points with the following two subroutines.

Subroutine I At each successor step, N reads at once a slice of $n + m + 3$ cells, replaces it with a new, equinumerous slice, and moves right or left to read a new slice. With subroutine I, we explain how to create a portion of a program that allows M to mimic the behavior of N. Suppose that, at a successor step α, for some $k \in \omega$, N reads the following slice of cells in the state q_0:

$$Sl = \big(i_0(k),\ \ldots,\ i_n(k),\ s_0(k),\ \ldots,\ s_n(k),\ o(k)\big).$$

Suppose that, after having read Sl, N replaces Sl with a new slice $(M, q_0)(Sl)$ and then moves left or right to read, at step $\alpha + 1$ and in state q_1, the new slice:

$$Sl' = \big(i_0(p),\ \ldots,\ i_n(p),\ s_0(p),\ \ldots,\ s_n(p),\ o(p)\big),$$

where $p = k \pm 1$. Since M starts with input $N'(r)$ and it has 0 everywhere on S and on O, we can assume that at step β

1. for every $0 \leq j \leq n$, the content of $i_j(k)$ appears in $I(k(n+1)+j)$
2. for every $0 \leq l \leq n$, the content of $s_l(k)$ appears on $S(k(n+1)+l)$
3. the content of $o(k)$ appears on $O(k(n+1))$.

In other words, we can assume that, at step β, M reads the slice:

$$\big(I(k(n+1)),\ S(k(n+1)),\ O(k(n+1))\big).$$

We now define a subroutine such that, for some $u \in \omega$, at step $\beta + u$, M will read the slice

$$\big(I(p(n+1)),\ S(p(n+1)),\ O(p(n+1))\big)$$

and the content of $i_j(k)$ will appear in $I(k(n+1)+j)$, the content of $s_l(k)$ will appear on $S(k(n+1)+l)$ and the content of $o(k)$ will appears on $O(k(n+1))$.

Let us now consider a slice of cells like Sl. We say that a sequence Q of $n+1$ triples of cells (z_j, x_j, y_j) is *equivalent* to Sl if

1. for every $0 \leq j \leq n$, $z_j = i_j(k)$
2. for every $0 \leq j \leq n$, $x_j = s_j(k)$
3. $y_0 = o(k)$
4. for every $1 \leq j \leq n$, $y_j = 0$.

To mimic the behavior of N at step α, M will do the following:

1. First, M reads the sequence of slices

$$\begin{gathered}\big(I(j(n+1)),\ S(j(n+1)),\ O(j(n+1))\big)\\ \big(I(j(n+1)+1),\ S(j(n+1)+1),\ 0\big)\\ \vdots\\ \big(I(j(n+1)+n),\ S(j(n+1)+n),\ 0\big).\end{gathered}$$

   which is equivalent to Sl. In this way, in $n+1$-many steps, M acquires the same amount of information that N receives in a single step.
2. Then, M replaces this sequence of slices with a new sequence of slices equivalent to $(M,\ q_0)(Sl)$.
3. Finally, M moves left or right $n+1$-many cells to read the sequence of slices equivalent to Sl'.

The details of the procedure are as follows. At step α, N carries out an instruction like the following

$$\mathscr{I} = (q_i,\ <z_0, \ldots, z_n, x_0, \ldots, x_n, y>,\ <z'_0, \ldots, z'_n, x'_0, \ldots, x'_n, y'>,\ R,\ q_j),$$

which means: if in the state q_i you see the sequence of cells $<z_0, \ldots, z_n, x_0, \ldots, x_n, y>$, then print $<z'_0, \ldots, z'_n, x'_0, \ldots, x'_n, y'>$, move right one cell, and enter state q_j. We choose new states $q_s, \ldots, q_{s+3n-1}$ and we write the following instructions:

1. $(q_i, {<}z_0, x_0, y{>}, {<}z_0, x_0, y{>}, R, q_s)$
2. $(q_s, {<}z_1, x_1, 0{>}, {<}z_1, x_1, 0{>}, R, q_{s+1})$
3. $(q_{s+1}, {<}z_2, x_2, 0{>}, {<}z_2, x_2, 0{>}, R, q_{s+2})$

   $\vdots$

4. $(q_{s+n-1}, {<}z_n, x_n, 0{>}, {<}z_n, x_n, 0{>}, L, q_{s+n})$
5. $(q_{s+n}, {<}z_{n-1}, x_{n-1}, 0{>}, {<}z_{n-1}, x_{n-1}, 0{>}, L, q_{s+n+1})$

   $\vdots$

6. $(q_{s+2n-1}, {<}z_0, x_0, y{>}, {<}z'_0, x'_0, y'{>}, R, q_{s+2n})$
7. $(q_{s+2n}, {<}z_1, x_1, 0{>}, {<}z'_1, x'_1, 0{>}, R, q_{s+2n+1})$

   $\vdots$

8. $(q_{s+3n-1}, {<}z_n, x_n, 0{>}, {<}z'_n, x'_n, 0{>}, R, q_j)$.

This list of instructions says that, if M finds a sequence of triples of cells that is equivalent to ${<}z_0, \ldots, z_n, x_0, \ldots, x_m, y{>}$, then M goes back to the beginning of the sequence and replaces it with a new sequence of slices that is equivalent to ${<}z'_0, \ldots, z'_n, x'_0, \ldots, x'_m, y'{>}$.

Subroutine II After completing subroutine I, the output tape O of M is such that, for every $x \in \omega$, $O((n+1)x) = o(x)$ and, for every $0 < i \leq n$, $O((n+1)x+i) = 0$. We define subroutine II to shift the content of $O((n+1)x)$ leftward so that, when M halts, its output tape is identical to the output tape of N when N halts. First, we print 0 everywhere on the scratch tape S of M. Then we give the following list of instructions to M:

1. Print 1 on $S(0)$.
2. Move right $n+1$ cells and read the triple (z, x, y). (Note that n is fixed, so we have enough states to implement this.)
3. Print $(z, 1, y)$.
4. If $y = 0$, then

   (5.i) move left until you see 1 on the scratch tape
   (5.ii) print 0 on the scratch tape
   (5.iii) move right one cell
   (5.iv) print 1 on the scratch tape
   (5.v) print 0 on the input tape
   (5.vi) move right until you find 1 on the scratch tape
   (5.vii) print 0 on the scratch tape
   (5.viii) move right $n+1$ cells
   (5.ix) go to instruction 3

5. If $y = 1$, then

   (6.i) print 0 on the output tape
   (6.ii) move left until you see 1 on the scratch tape
   (6.iii) print 0 on the scratch tape
   (6.iv) move right one cell
   (6.v) print 1 on the scratch tape
   (6.vi) print 1 on the input tape

(6.vii) move right until you find 1 on the scratch tape
(6.viii) print 0 on the scratch tape
(6.ix) move right $n + 1$ cells
(6.x) go to instruction 3

In other words, the above list of instructions tells M to check the content of every cell $O((n + 1)x)$ and to shift that content to cell $O(x)$. We use the scratch tape to keep track of the cells $O((n + 1)x)$ that have already been examined and of the cell on the output tape that one should use to record the content of $O((n + 1)x)$. □

From Theorem 14 and from the observation that a wITTM$_{<\omega,<\omega}$ can compute every function that a wITTM can compute, it follows that, modulo coding, the wITTM-p.c. functions are exactly the wITTM$_{<\omega,<\omega}$-p.c. functions. We record this piece of information in the following theorem.

Equivalence Theorem 15 *Modulo coding, the wITTM-p.c. functions are exactly the functions that are wITTM$_{<\omega,<\omega}$-p.c.*

We can also show that, for every function f, the computation of f by a wITTM$_{<\omega,<\omega}$ is not significantly faster than the computation of f (modulo coding) by a wITTM. We make this observation precise in the following theorem.

Theorem 16 *Let r, N, and N' as in Theorem 14. Suppose that $N(r)$ halts at step α. Then the following statements are true:*

1. *If α is a limit ordinal, then, for some wITTM M, $M(N'(r))$ halts at step α.*
2. *If α is either 0 or a successor ordinal, then, for some wITTM M, for some $k \in \omega$, $M(N'(r))$ halts at step $\alpha + k$.*

Before proving this statement, it is useful to consider once again the behavior of M as defined in the proof of Theorem 14. For every successor step of N, M carries out a greater, but finite number of steps to produce a sequence of triples equivalent to the slice produced by N and to reach the right position for continuing the computation. After having done this, M cleans the scratch tape (which requires ω-many steps) and then shifts the content of the output tape to the left (which requires ω-many more steps). One can interleave the operation of cleaning the scratch tape and the operation of shifting the content on the output tape to the left so that M completes both operations in just ω-many steps. Therefore, if α is a successor ordinal, then M stops in $\alpha + \omega$ steps. One can see that the same is true if α is 0 or a limit ordinal. In the following proof of Theorem 16, we define a different program for M so that Theorem 16 holds.

Proof We define M so that it carries out the following operations:

1. M replaces the input $N'(r)$ with the following sequence

$$r_0(0)r_1(0)\ldots r_n(0)Ar_0(1)r_1(1)\ldots r_n(1)Ar_0(2)\ldots r_0(m)r_1(m)\ldots r_n(m)A\ldots,$$

   where A is a new tape symbol different from 0 and 1. In other words, we insert the symbol A between each $r_n(j)$ and $r_0(j + 1)$ in $N'(r)$. We call this process subroutine III.

2. In the proof of Theorem 14, M represented a slice like Sl of N by a series of triples

$$\begin{gathered}\big(I(k(n+1)),\ S(k(n+1)),\ O(k(n+1))\big)\\ \big(I(k(n+1)+1),\ S(k(n+1)+1),\ 0\big)\\ \vdots\\ \big(I(k(n+1)+n),\ S(k(n+1)+n),\ 0\big).\end{gathered}$$

   This is the type of representation of slices of cells that we used in the proof of Theorem 14. We refer to these representations as representations of type 1. We now define a new way to represent the same slice Sl of M by the following series of triples:

$$\begin{gathered}\big(I(k(n+2)),\ S(k(n+2)),\ 0\big)\\ \big(I(k(n+2)+1),\ S(k(n+2)+1),\ 0\big)\\ \vdots\\ \big(I(k(n+2)+n),\ S(k(n+2)+n),\ 0\big).\\ \big(A,\ S(k(n+2)+n+1),\ 0\big),\end{gathered}$$

   where $S(k(n+2)+n+1)$ is defined to contain the same value that $o(k)$ of N contains. We say that representations of this new type are representations of type 2. Series of both types can be used to acquire the same information that N acquires at once by reading the slice Sl. Concerning the content that is relevant here, Sl, its representation of type 1, and its representation of type 2 are all informationally equivalent.
3. As explained in the proof of Theorem 14, using subroutine I, M acquired the information that N received from Sl by reading a series of triples of type 1. Moreover, M recorded the information that N recorded in Sl' by printing a new series of triples of type 1. Now, we modify subroutine I, so that M acquires the information that N receives from Sl by a series of triples of type 2. Then, M produces a new series of triples to record the same information that N records by printing the slice Sl' by printing a series of triples of type 2. In other words, when N carries out an instruction like $\mathscr{I}$ in the proof of Theorem 14, now M first reads a series of triples of type 2 acquiring the same information that N received. Then, if required, M prints a series of triples of type 2 that contains the same information that N produced.
4. We then define a new process, which we call subroutine IV. Once M has mimicked the behavior of N carrying out an instruction like $\mathscr{I}$, before mimicking what N does next, M copies the content in $S(k(n+2)+n+1)$ to $O(k)$. To copy the content in $S(k(n+2)+n+1)$ to $O(k)$, M works in the following way:
   (a) Let us suppose to have at disposal the already mentioned symbols 0, 1, and A and the new tape symbols B, C, D, E, and F all different from one another.

(b) Let us call $\mathscr{J}$ the instruction that N carries out immediately before carrying out $\mathscr{I}$. We assume that, mimicking N that carries out $\mathscr{J}$, M has done the following:
   i. M printed the symbol B on cell $O(k\pm1)$ if N, carrying out $\mathscr{J}$ immediately before $\mathscr{I}$ printed 0 on $o(k \pm 1)$ and then moved right.
   ii. M printed the symbol C on cell $O(k\pm1)$ if N, carrying out $\mathscr{J}$ immediately before $\mathscr{I}$ printed 0 on $o(k \pm 1)$ and then moved left.
   iii. M printed the symbol D on cell $O(k \pm 1)$ if N, carrying out $\mathscr{J}$ immediately before $\mathscr{I}$ printed 1 on $o(k \pm 1)$ and then moved right.
   iv. M printed the symbol E on cell $O(k\pm1)$ if N, carrying out $\mathscr{J}$ immediately before $\mathscr{I}$ printed 1 on $o(k \pm 1)$ and then moved left.

(c) Now, M, being over $S(k(n + 2) + n + 1)$, prints F on $I(k(n + 2) + n + 1)$.

(d) Then, M looks for one among the symbols B, C, D, or E. To find it, first M goes left. If it does not find one of these symbols, M goes right until it finds it.

(e) When M sees either B or C, M replaces it with 0 and, when M sees either D or E, M replaces it with 1.

(f) Then, M moves right one cell (if it sees either B or D) or left once cell (if it sees either C or E) and does the following:
   i. M prints B if $\mathscr{I}$ requires N to print 0 and move right;
   ii. M prints C if $\mathscr{I}$ requires N to print 0 and move left;
   iii. M prints D if $\mathscr{I}$ requires N to print 1 and move right;
   iv. M prints E if $\mathscr{I}$ requires N to print 1 and move left.

(g) Then, M looks for F on the input tape and moves right one cell or left $2n + 3$ cells as required to mimic the next step of N's computation (if any).

One observes that the content of no cell of M changes infinitely often if the content of no cell of N does so.

In plainer words, using subroutine III, M first re-arranges the content on the input tape so that there is an extra cell at the end of each sequence $r_0(k), \ldots, r_n(k)$ on the input tape. Then M uses the modified subroutine I to mimic the behavior of N at a certain successor step. In the proof of Theorem 14, M first finished simulating the computation of N and then prepared its output tape so that it became identical to the output tape of N. Now, using subroutine IV, M takes care of arranging the output tape appropriately before simulating what N does at the next step. We no longer use subroutine II, which we defined in the proof of Theorem 14. Subroutine III is used only between step 0 and step ω. By interleaving the three subroutines, M can prepare enough of the input tape, simulate the behavior of N at the proper step, and record the result on the appropriate cell on the output tape in finitely many steps. □

Finally, one can observe that the results on the halting time of wITTMs proved in Sect. 2.3 also hold for wITTM$_{<\omega,<\omega}$s.

3.2 Finitely many tapes and heads

In this section, we define a new variant of wITTMs. In the previous section, we considered wITTMs with possibly finitely many extra input and scratch tapes, but

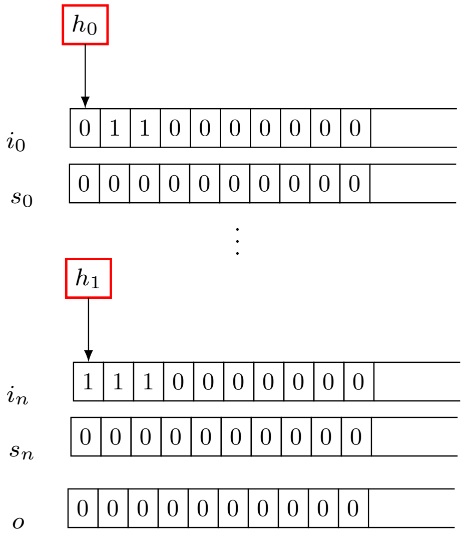

Fig. 4 Weak infinite time Turing machine with finitely many extra input tapes, scratch tapes, and read–write heads

always with a single read–write head. Now we consider wITTMs with possibly finitely many extra input and scratch tapes and also read–write heads. More precisely, we consider machines that look like the one represented in Fig. 4. Let us write wITTM$_{<\omega}$ to refer to a machine of this type. Each wITTM$_{<\omega}$ consists of n-many pairs (i_j, s_j) of input and scratch tape. Each pair has its read–write head h_j. At each step, h_j reads the triple of cells $(i_j(k), s_j(k), o(k))$ and, as required by the program, prints a new triple, moves right or left, or stops. Each head starts at the same instant 0. However, different heads can have different programs and can halt at different times. The computation of a wITTM$_{<\omega}$ ends when (and if) every head has halted.

Different heads read and print on different input and scratch tapes. However, they all read and print on the same output tape. Different heads can print different symbols at the same time on different cells of the output tape. They can also print the same symbol at the same time on the same cell of the output tape. However, they cannot print different symbols at the same time on the same cell of the output tape. If the program requires them to do so, the machine hangs: the computation does not proceed, and no output is given. We consider wITTM$_{<\omega}$s to compute functions $f : (2^\omega)^{<\omega} \to 2^\omega$. Given an input $r = (r_0, \ldots, r_n)$, where each $r_j \in 2^\omega$, and a wITTM$_{<\omega}$ N with exactly $n + 1$ heads, we write r_j on i_j.

Figure 4 provides an example of a machine of this type. Let us call this machine N. N has two heads, h_0 and h_1. Let us suppose that the program for h_0 contains the following instructions:

1. $\big(q_0,\ <0,0,0>,\ <0,1,0>,\ R,\ q_0\big)$
2. $\big(q_0,\ <1,0,0>,\ <1,1,0>,\ R,\ q_0\big)$.

Then, let us suppose that the program for h_1 contains the following two instructions:

1. $\big(q_0,\ <1,0,0>,\ <1,1,0>,\ R,\ q_1\big)$
2. $\big(q_1,\ <1,0,0>,\ <0,1,0>,\ L,\ q_0\big)$.

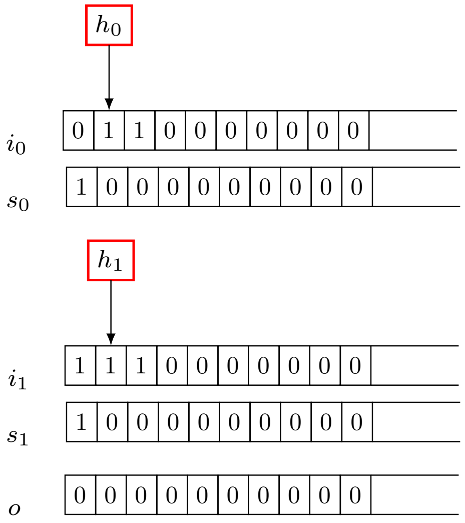

Fig. 5 The wITTM$_{<\omega}$ N at step 1

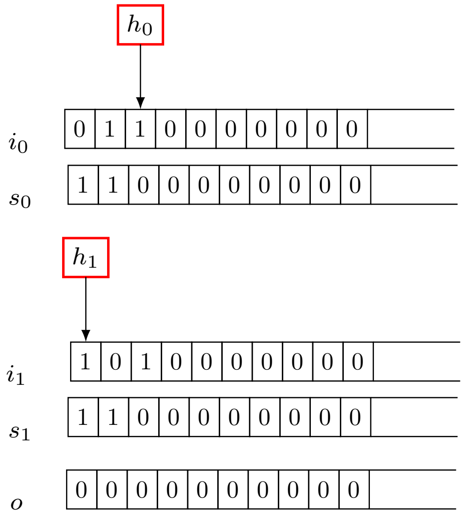

Fig. 6 The wITTM$_{<\omega}$ N at step 2

Let us take q_0 as the initial state of both heads. Figures 4, 5 and 6 represent the first three steps of the computation of N.

One can regard wITTM$_{<\omega,<\omega}$s as a particular type of wITTM$_{<\omega}$s, i.e those wITTM$_{<\omega}$s whose heads are all always in the same state, move in the same direction, and print the same symbol on the output tape. With a little abuse of notation, one can write

$$\text{wITTM} \subseteq \text{wITTM}_{<\omega,<\omega} \subseteq \text{wITTM}_{<\omega}.$$

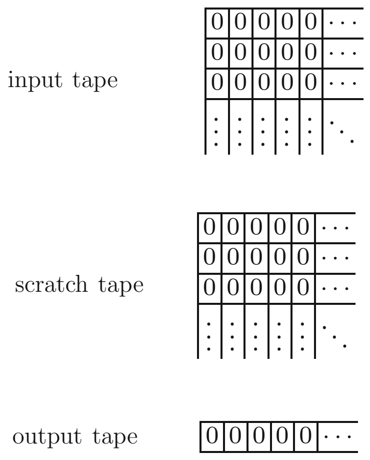

Fig. 7 Weak infinite time Turing machine with bidimensional input and scratch tapes

We have seen already that wITTMs and wITTM$_{<\omega,<\omega}$s have the same computational power. The above containment relation, shows that everything that is wITTM$_{<\omega,<\omega}$-computable is also wITTM$_{<\omega}$-computable.

3.3 Bidimensional tapes and single head

The last variant of weak infinite time Turing machines that we consider comprises machines each of which has a bidimensional input tape, a bidimensional scratch tape, and a one-dimensional output tape, as in Fig. 7. We consider the head of a machine of this type to consist of three parts, each reading one cell on, respectively, the input tape, the scratch tape, and the output tape. The part on the input tape and the part on the scratch tape can move left, right, up, and down one cell. At the beginning of the computation they read the top left cell of their respective tape. During the computation each head-part can move independently of the other parts. The following is an example of an instruction for such a machine:

$$\big(q,\ <x, y, z>,\ <x', y', z'>,\ <U, R, L>,\ p\big),$$

which says: if the machine is in state q and sees $< x, y, z >$, then

- the machine writes $<x', y', z'>$
- the head-part on the input tape moves up one cell
- the head-part on the scratch tape moves right one cell
- the head-part on the output tape moves left one cell
- the machine enters into state p.

At limit steps, the head-parts go back to their initial positions. We refer to a machine of this type as wITTM$_{\omega,\omega}$.

We write i_k to refer to the row number k on the input tape. At the beginning of a computation, every i_k has a real written on it. All the other cells contain 0s. Given a sequence of reals $(r_k)_{k\in\omega}$, we say that a computation starts on input $(r_k)_{k\in\omega}$ if r_k

appears on i_k at step 0. We prove that, in the sense specified by Theorem 17 and Lemma 19, wITTM$_{\omega,\omega}$s and wITTM$_{<\omega}$s are computationally equivalent. Given a finite sequence of reals $r = (r_0, \dots, r_n)$, we say that $r' = (r'_k)_{k\in\omega}$ is a *completion* of r if, for $0 \leq j \leq n$, $r_j = r'_j$ and, for $j > n$, $r'_j = 0$. As before, we write $M(r) \downarrow$ to mean that the machine M converges on input r.

Theorem 17 *For every $n \in \omega$, for every wITTM$_{<\omega}$ M, for some wITTM$_{\omega,\omega}$ N, for every $r \in (2^\omega)^n$, if r' is a completion of r, then*

1. *if $M(r) \downarrow$, then $N(r') = M(r)$ and*
2. *if $M(r) \uparrow$, then $N(r') \uparrow$.*

Proof Suppose that M has exactly $k + 1$ pairs of input and scratch tapes. We write i_j for the row number j on the input tape of N and s_j to refer to the row number j on the scratch tape of N. At the beginning of N's computation, r'_j is written on i_j. Therefore, for $j > k$, i_j contains 0. We select a new tape symbol F. We write F everywhere on i_{k+1}. We do the same with s_{k+1}. The idea is that, for $0 \leq j \leq k$, we use i_j and s_j to reproduce what M writes on its input tape and scratch tape number j respectively. For $j > k$, we use the infinitely many rows i_j and s_j to keep track of the positions of the various heads of M whose work we want to mimic, paying attention not to change the content of any cell infinitely often.

Let us suppose, for simplicity, that M has exactly two pairs of input and scratch tapes. Let A be the head reading the first pair and B the head reading the second pair. Let us suppose that, at step t, A prints 1 on cell n of its input tape and moves right. Moreover, let us suppose that B prints 0 on cell m of its input tape and moves left. We use i_0 to reproduce the content of input tape A of M and i_1 to reproduce the content of input tape B of M. The head-part $h(i)$ of N on the input tape behaves in the following way:

1. First, $h(i)$ uses subroutine S_1 (which we will define below) to find cell n on i_0.
2. Then $h(i)$ prints 1.
3. Then $h(i)$ uses subroutine S_2 (which we will define below) to record the next cell that it wants to scan to mimic the behavior of A (i.e. it records that it wants to scan cell $n + 1$ on i_0).
4. Then $h(i)$ uses subroutine S_3 (which we will define below) to find the cell to be scanned on i_1 to mimic the behavior of B, i.e. it finds cell m on i_1.
5. Then $h(i)$ prints 0 on cell m of i_1.
6. Then $h(i)$ uses subroutine S_4 to record which cell it wants to scan next on i_1 to mimic the behavior of B, i.e. it records that it wants to scan cell $m - 1$ on i_1.
7. Then $h(i)$ uses subroutine S_1 to find the cell to be scanned on i_0 to mimic the behavior of A, i.e. it finds cell $n + 1$ on i_0.

Repeating this procedure, $h(i)$ updates the content of i_0 and i_1, as A and B update the content of their respective input tapes. Now we define the subroutines mentioned above.

Subroutine 1 $h(i)$ uses this subroutine to determine which cell on i_0 it should read to mimic the behavior of the head A of M. To keep track of cell c on i_0 that $h(i)$ should read at this point, we write a new tape symbol G on the same column where c is. We

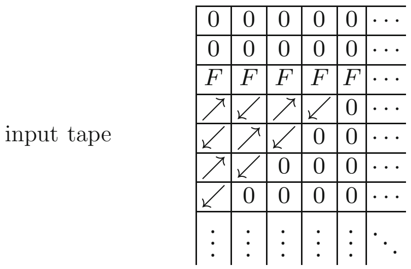

Fig. 8 Diagonal movements of $h(i)$ in subroutine S_1

use the fact that the input tape has infinitely many rows to avoid writing on the same cell infinitely often. The procedure is as follows:

1. First, $h(i)$ looks for the leftmost F on the input tape.
2. Then, $h(i)$ scans the cell immediately under the leftmost F.
3. Then $h(i)$ moves along increasingly long diagonals moving alternatively upward and downward. Figure 8 shows how $h(i)$ moves. We may imagine having an extra symbol to keep track of which leftmost cell should be examined next.
4. When $h(i)$ finds the new tape symbol G, it prints C.
5. Then $h(i)$ reads the content of the cell in the first row and in the same column where G was written.

Subroutine 2 we use this subroutine to record which cell on i_0 we should later scan when we resume mimicking the behavior of the head A of M. The procedure is as follows:

1. When $h(i)$ finishes working on i_0, it searches for the first cell c that contains 0 that is on the same column and below F.
2. Then, $h(i)$ prints G on c.

Subroutine 3 similar to subroutine 1, but we use it to find the new tape symbol H that indicates which cell on tape i_1 we should now scan to mimic the behavior of the head B of M.

Subroutine 4 similar to subroutine 2, but we use it to write the new tape symbol H that indicates which cell on tape i_1 we should scan when we resume mimicking the behavior of the head B of M.

One then adds a similar procedure to mimic the behavior of M on its various scratch tapes using N's bidimensional scratch tape.

To complete the proof we now explain how $h(o)$, the head reading and printing on the output tape of N, mimics the behavior of A and B when they write on the output tape of M. To find the cell that A reads and on which it possibly writes, $h(o)$ behaves as follows. When $h(i)$ records the position of A on the output tape, $h(i)$ and $h(o)$ go to the leftmost cell ($h(i)$ does not change row). Then $h(i)$ and $h(o)$ move right together until $h(i)$ recognizes the cell on which A works at the step of the computation that N is currently mimicking. Behaving similarly, $h(o)$ finds the cell corresponding to the cell that B reads and on which it possibly writes at that step of the computation.

Now, we explain how N understands if M attempts to write two different symbols at the same time on the same cell of the output tape. To do that, we describe the following subroutine for N:

1. Suppose that A print x (either 0 or 1) on the output tape of M on cell n. Then $h(o)$ will print X on its output tape if $x = 0$ and Y otherwise.
2. Then $h(o)$ mimics the behavior of B. We can have the following two cases.
   (a) *Case* 1 $h(o)$, mimicking B, attempts to write either 0 or 1 on a cell containing either 0 or 1.
      i. In this case, $h(o)$ completes its operation.
      ii. Then, $h(o)$ moves left and later, if needed, also right to look for either X or Y. if $h(o)$ finds X, it replaces it with 0. If $h(o)$ finds Y, it replaces it with 1.
      iii. Then N resume mimicking the next step of the computation of M applying the procedures described above.
   (b) *Case* 2 $h(o)$ attempts to write 0 on a cell containing Y or 1 on a cell containing X.
      i. This means that, at the corresponding step in the computation of M, A and B try to write two different symbols on the same cell of the output tape of M. Therefore, M hangs at this step.
      ii. To mimic the behavior of M, N has to hang. To do that, we suppose that $h(o)$ enters a loop in which it prints alternatively 0 and 1 on the same cell without doing anything else. In this way, N hangs at the next limit step. □

In the proof of Theorem 17 we can interleave the process of writing F on i_{k+1} and G on o_{k+1} with subroutines 1–4. One then notices that, when N mimics the computation of M, N completes its computation using (a) the same number of limit steps as M if N never attempts to write different symbols on the same cell of the output tape at the same time and (b) at most one extra limit step otherwise. Therefore, in the sense specified by the following Lemma, the computation of N that mimics M is not significantly slower than the computation that M carries out.

Lemma 18 *Let r, r', M, and N as in Theorem 17. Suppose that $M(r)$ halts at step α. Then the following statements are true:*

1. *If α is a limit ordinal, then $N(r')$ halts at step α.*
2. *If α is either 0 or a successor ordinal, then, $N(r')$ halts at or before the next limit step.*

Proof The proof is clear from the procedure described in the proof of Theorem 17. The only thing that one should change is the procedure for writing a row containing F everywhere that separates the row used to reproduce the input tapes of M and the rows used for subroutines 1–4. One interleaves the process of writing such a row with the process of mimicking the computation of N to avoid taking one extra limit step to complete the computation. □

Lemma 19 *For some wITTM$_{<\omega}$ M, for some wITTM$_{\omega,\omega}$ N', for every wITTM$_{\omega,\omega}$ N, for every sequence $(r_k)_{k\in\omega}$ of reals, $N'\big((r_k)_{k\in\omega}\big) \in 2^\omega$ and*

1. *if $N(r)\downarrow$, then $M\big(N'((r_k)_{k\in\omega})\big) = N(r)$ and*
2. *if $N(r)\uparrow$, then $M\big(N'((r_k)_{k\in\omega})\big)\uparrow$.*

Proof Given a sequence $(r_k)_{k\in\omega}$ of reals, N' produces a new real $f \in 2^\omega$ such that

$$r = r_0(0)r_0(1)r_1(0)r_0(2)r_1(1)r_2(0)\ldots$$

where $r_i(j)$ is the digit number j in the real number i. In other words, we define the usual pairing function $p(x, y) = (x^2 + 2xy + y^2 + 3x + y)2^{-1}$ and the functions $p_0(p(x, y)) = x$ and $p_1(p(x, y)) = y$. We use these functions to describe the real $r(n) = r_{p_0(n)}(p_1(n))$, which codes the infinitely many reals in $(r_k)_{k\in\omega}$. Then we consider a wITTM$_{<\omega}$ M with four pairs of input and scratch tapes, which we call A, B, C, and D. We will refer to the input tape and the scratch tape of A as i_A and s_A respectively and similarly for the tapes of B, C, and D. We put r on i_A. The idea of the proof is to use i_A to keep track of the content of the input tape of N and s_A to keep track of the scratch tape of N. We use B, C, and D to keep track of the movement of the three parts $h(i)$, $h(s)$, and $h(o)$ of N's head. For example, suppose that M prints 0 on cell $p(i,\ j)$ to mimic the behavior of N that, in the corresponding circumstances, prints 0 on cell number j of row number i on the input tape. Suppose that, then, N moves right one cell. Then M will use the head on B to compute $p(i,\ j+1)$. To avoid rewriting on the same cell infinitely often, B does the following:

1. It writes the number $p(i,\ j + 1)$ as a string of $p(i,\ j + 1) + 1$ 1s.
2. It prints a new tape symbol F immediately to the right of the last 1 printed according to the previous instruction.

The next time B needs to compute the position to which A should go to mimic the behavior of $h(i)$, B does the following:

1. First, B looks for F, then replaces every 1 to the left of F with 0.
2. Then, B goes to the right of F and writes a new string of 1s as required.
3. Then, B writes F to the right of the new string of 1s.
4. Then, B goes left and replaces the old F with 0.

In this way, B always uses a new portion of the tape and never writes in the same cell unboundedly often between one limit step (or the initial step) and the following limit step. Moreover, at every limit step, the tapes in B have been cleaned completely (i.e., they contain 0 in every cell) and are ready for being used again in the way described above. The only thing that still requires explanation is how A can find the right position to work on the input tape once B has written the relevant string of 1s somewhere on its input tape. The solution is that A and B can move at the same time while A starts from the leftmost cell and B starts from the second 1 in the list. Once B reaches F, then A knows that it has found the right cell. Similarly, we use C to compute the position on A's scratch tape to which A goes to mimic the behavior of $h(s)$. We also use D in a similar way to tell A which cell on the output tape it should scan at a particular step. □

From Theorem 17 and Lemma 19, it follows that, modulo coding, the wITTM$_{<\omega}$-p.c. functions are exactly the wITTM$_{\omega,\omega}$-p.c. functions. We record this piece of information in the following theorem.

Equivalence Theorem 20 *Modulo coding, the functions that are wITTM$_{<\omega}$-p.c. are exactly the wITTM$_{\omega,\omega}$-p.c. functions.*

The next result follows from the proof of Lemma 19.

Theorem 21 *Let $(r_k)_{k\in\omega}$, M, N, and N, as in Lemma 19. Suppose that $N(r)$ halts at step α. Then the following statements are true:*

1. *If α is a limit ordinal, then $M\big(N'((r_k)_{k\in\omega})\big)$ halts at step α.*
2. *If α is either 0 or a successor ordinal, then, for some $k \in \omega$, $M\big(N'(r_k)_{k\in\omega})\big)$ halts at step $\alpha + k$.*

4 Arithmetic hierarchy and computational strength

We have now introduced weak infinite time Turing machines of four different types: wITTMs, wITTM$_{<\omega,<\omega}$s, wITTM$_{<\omega}$s, and wITTM$_{\omega,\omega}$s. In Sect. 2.3 we showed a series of results on the halting time of wITTMs. All these results readily apply (with the same proofs) to all the variants that we have considered. In particular, we note the following piece of information.

Halting Time Theorem 22 *ω^2 is the set of all halting times of wITTM$_{\omega,\omega}$s.*

In Equivalence Theorem 15, we showed that wITTMs and wITTM$_{<\omega,<\omega}$s are computationally equivalent: the wITTM-p.c. functions are exactly the wITTM$_{<\omega,<\omega}$-p.c. functions (modulo coding of the input). In Equivalence Theorem 20, we showed that wITTM$_{<\omega}$s and wITTM$_{\omega,\omega}$s are also computationally equivalent. It is clear that a wITTM$_{\omega,\omega}$ can compute everything that a wITTM can compute. We also observed that wITTMs are computationally more powerful than regular Turing machines. We will later show that wITTM$_{\omega,\omega}$s are computationally weaker than regular infinite time Turing machines. Introducing a new piece of notation, we summarize these observations as follows:

$$\text{TM} < \text{wITTM} \equiv \text{wITTM}_{<\omega,<\omega} \leq \text{wITTM}_{<\omega} \equiv \text{wITTM}_{\omega,\omega} < \text{ITTM}.$$

In sum, we have described four models of infinitary computation whose strength lies strictly between Turing machines and ITTMs. As far as we have shown, these four models individuate two classes of computable functions over the reals: the wITTM-p.c. functions and the wITTM$_{\omega,\omega}$-p.c. functions. Are these two classes the same? In other words, are wITTMs computationally equivalent to wITTM$_{\omega,\omega}$s? We are unable to answer this question. However, in this section we provide further information about the strength of these models of computation. In particular, we note the following:

1. We show that the wITTM-decidable relations on natural numbers and the wITTM$_{\omega,\omega}$-decidable relations on natural numbers are exactly the arithmetic relations.
2. We define the arithmetical hierarchy of relations on natural and real numbers. We show that there are relations on natural and real numbers that are both wITTM-computable and non-arithmetical.
3. There are wITTM-semi-decidable sets that are not wITTM-decidable. Similarly, there are wITTM$_{\omega,\omega}$-semi-decidable sets that are not wITTM$_{\omega,\omega}$-decidable.
4. The halting set for wITTMs and the halting set for wITTM$_{\omega,\omega}$s are ITTM-decidable.

4.1 Arithmetic relations

We define the arithmetical hierarchy of relations on real and natural numbers. Hartley Rogers discussed a similar notion in greater detail in his book on computability (see [6] §15.2), where he considers relations on natural numbers and functions over the natural numbers. A brief presentation is the following. Let $x_1,\ x_2,\ \dots$ be variables for natural numbers and $r_1,\ r_2,\ \dots$ be variables for reals in 2^ω. We write $\bar{x}$ for a finite sequence of variables of the first type and $\bar{r}$ for a finite sequence of variables of the second type. A relation $R(\bar{x}, \bar{r})$ is Turing computable if and only if, for some Turing program $e, x_1,\ \dots,\ x_{|\bar{x}|} \in \omega$, for every $r_1,\ \dots,\ r_{|\bar{r}|} \in 2^\omega$,

$$\varphi_e^{\bar{r}}(\bar{x}) = \begin{cases} 1 & \text{if } R(\bar{x}, \bar{r}) \text{ holds} \\ 0 & \text{otherwise.} \end{cases}$$

We say that a relation $R(\bar{x}, \bar{r})$ is Σ_0^0 ($= \Pi_0^0$) if and only if it is Turing computable. Then we define:

- $R(\bar{x},\ \bar{r})$ is Σ_{n+1}^0 if and only if, for some relation $P(\bar{x},\ y,\ \bar{r}) \in \Pi_n^0$, $R(\bar{x},\ \bar{r}) \Leftrightarrow \exists y P(\bar{x}, y, \bar{r})$
- $R(\bar{x},\ \bar{r})$ is Π_{n+1}^0 if and only if, for some relation $P(\bar{x},\ y,\ \bar{r}) \in \Sigma_n^0$, $R(\bar{x},\ \bar{r}) \Leftrightarrow \forall y P(\bar{x},\ y,\ \bar{r})$.

Definition 23 Let $R(\bar{x}, \bar{r})$ be a relation of natural and real numbers. We say that R is wITTM-*writable* if, for some wITTM M, starting with $\bar{r}$ on the input tape, for every $\bar{x}$, M halts with 1 on cell number $p(\bar{x})$ of the output tape, where $p(\bar{x})$ codes $\bar{x}$, and 0 everywhere else. We define wITTM$_{\omega,\omega}$-writable relations in an analogous way.

We can then prove Theorems 24 and 26.[4]

Theorem 24 *If a relation of natural and real numbers is wITTM$_{\omega,\omega}$-writable, then it is arithmetical relative to $\bar{r}$.*

Proof Without loss of generality, we consider only relations $R(\bar{x}, r)$ of natural numbers $\bar{x}$ and one real r. We show that the content of the output tape of a wITTM$_{\omega,\omega}$ M at or before step ωk is Turing computable from $r^{(k)}$. This is true for $k = 1$, since a wITTM$_{\omega,\omega}$-computation of length strictly less than ω is a regular Turing computation and a wITTM$_{\omega,\omega}$-computation of length ω is a Turing computation in the limit. Then, let us assume that the real c describes the configuration of the machine at step ωk. One can describe the content of the output tape at step $\omega(k + 1)$ from c' in the following way. Define a program N such that, with oracle c, for every i and $j \in \omega$, $N(i,\ j)$ halts if and only if the content of the cell i on the output tape of M does not change after step

[4] In his forthcoming book *Ordinal computability. An introduction to infinitary machines*, Merlin Carl proves the following statements:

1. If $S \subseteq \omega$ is wITTM-writable, then, for some n, $S \leq_T \emptyset^{(n)}$.
2. For every $S \subseteq \omega$, for some wITTM M, M on input S outputs S'.

Carl's proofs of these statements essentially work also as proofs of, respectively, Theorems 24 and 26. We slightly adapt Carl's ideas to provide the proofs in this article.

j. Such a j exists, otherwise the computation of M would be undefined. Therefore, to know the content of cell i of M's output tape at step $\omega(k+1)$, it is enough to consider the content of that cell at step $\omega k + j$ of M's computation. Given c', we can determine whether or not $N^c(i, j)$ halts. Therefore, the content of the output tape of M at or before step $\omega(k+1)$ is Turing computable in c'. By the induction hypothesis, we have $c \leq_T r^{(k)}$. Therefore, $c' \leq_T r^{(k+1)}$. We conclude that the content of the output tape of M at or before step $\omega(k+1)$ is Turing computable in $r^{(k+1)}$ and, therefore, arithmetical in r. $\square$

The above proof shows that, for each real r, there is a natural number k such that the set $\{\bar{x} \in \omega : R(\bar{x}, r)\}$ is computable from $r^{(k)}$. However, k might depend on r. Instead of considering relations on real and natural numbers, let us consider for a moment only relations on natural numbers. The above proof shows that every relation on natural numbers that is wITTM$_{<\omega,<\omega}$-writable is arithmetical. One can sharpen these observations as follows.[5]

Theorem 25 *For some relation of natural and real numbers R, R is wITTM$_{<\omega,<\omega}$-writable but it is not arithmetical.*

Proof Let φ_e^s be a listing of all Σ_n^0 relations on real and natural numbers. Define the relation $R(x, r)$ such that, for all $x \in \omega$, $R(x, 0) = 0$ and, if r begins with $0^{p(e,s)}1$, then $R(x, r) = 1 - \varphi_e^s(x, r)$. Notice that, for $r = 0^{p(e,s)}10^\infty$, $\{x \in \omega : R(x, r)\}$ is not Σ_n^0 relative to r, since R diagonalizes against all Σ_n^0 sets. We conclude that, for every $n \in \omega$, there is a real r such that $\{x \in \omega : R(x, r)\}$ is not Σ_n^0 relative to r. Thus, R is not arithmetical. $\square$

Theorem 26 *If a relation of natural and real numbers is arithmetical, then it is wITTM-writable.*

Proof Without loss of generality, let us consider only relations $R(\bar{x}, r)$ of natural numbers $\bar{x}$ and one real r. A Turing machine with oracle $r^{(n)}$ can decide every Σ_n^0-relation. The idea for the proof is to define a wITTM that, given an arithmetic relation $R(\bar{x}, r)$, computes enough jumps of r so that it will then decide whether $R(\bar{x}, r)$ holds as a regular Turing machine. After that, it will set $o\big(p(\bar{x})\big) = 1$ if and only if $R(\bar{x}, r)$ holds. We compute $r^{(n+1)}$ from $r^{(n)}$ starting from r. The only issue is to avoid writing in the same cell infinitely often. The details are the following. Suppose that $r^{(n)}$ is on the input tape. We organize the computation in phases.

1. At the beginning of phase number m the head will be on the unique triple of cells whose intermediate cell (i.e. the one on the scratch tape) contains the symbol F. The portion of the scratch tape immediately to the right of F contains the sequence 1^m, representing the number m.
2. Using the portion of the scratch tape further to the right of 1^m, we compute $p_0(m)$ and $p_1(m)$, where p_0 and p_1 are as in the proof of Lemma 19. At the end of this subroutine, the portion of the scratch tape to the right of F looks like this: $1^m X 1^{p_0(m)} Y 1^{p_1(m)}$ where X and Y are new tape symbols.

[5] In a personal conversation with me, Dan Turetsky communicated the following observation together with its proof.

3. Then we compute $\varphi^{r^{(n)}}(p_0(m))$ for exactly $p_1(m)$ many steps. We can have two cases.
   (a) Suppose that $\varphi^{r^{(n)}}(p_0(m))$ halts in exactly $p_1(m)$ many steps. In this case, we use the sequence $1^{p_0(m)}$ between X and Y on the scratch tape to find cell number $p_0(m)$ on the output tape and we print 1.
   (b) Suppose that $\varphi^{r^{(n)}}(p_0(m))$ does not halt in exactly $p_1(m)$ many steps. In this case, we move to the next step.
4. Then we look for F on the scratch tape. We replace it by 0 and we move to the right. When we find 0, we print F, and we move to phase $m+1$ of the computation.

Following this procedure, at the next limit step, $r^{(n+1)}$ is on the output tape. If we need to compute more jumps, we can copy the content of the output tape onto the input tape. Then we clean the scratch tape and the output tape and repeat the procedure described above. Interleaving the various procedures, we obtain the desired $r^{(n)}$ using exactly n many limit steps. Therefore, we do not change any content on the input tape infinitely often. For the same reason, the same holds for the content on the output tape. The process of using portions of the scratch tape that are more and more to the right ensures that we do not change any content on the scratch tape infinitely often. Once we obtain the desired $r^{(n)}$ we can decide $R(\bar{x}, r)$ as a regular Turing machine. Once we have decided $R(\bar{x}, r)$, we set $o\big(p(\bar{x})\big) = 1$ if and only if $R(\bar{x}, r)$ holds. □

The next result follows from Theorems 24 and 26.

Theorem 27 *The wITTM$_{\omega,\omega}$-writable relations on natural numbers are exactly the wITTM-writable relations on natural numbers, i.e. the arithmetic relations.*

In [5] (Theorem 2.6), Hamkins and Lewis showed that the sets of natural numbers that are ITTM-decidable with time bound ω^2 are exactly the arithmetic sets. From Theorem 27, we can infer that the same is true for wITTM-decidable sets too. In other words, the rule *Lim Sup* is unnecessary for deciding all and only the arithmetic relations on natural numbers. The rule *Eventually Constant* is sufficient.

Theorem 28 *The wITTM-decidable relations on natural numbers are exactly the arithmetic relations.*

The rule *Eventually Constant* appears to be the minimal meaningful modification necessary to extend Turing's model of computation into transfinite ordinal time. In this sense, as one takes Turing machines to characterize the Δ^0_1 relations on natural numbers because these are exactly the relations on natural numbers that are Turing computable, one can take wITTMs to characterize the arithmetic relations on natural numbers.

4.2 Halting sets

We state the following theorem.

Theorem 29 *For every relation on natural numbers R, R is wITTM-decidable if and only if R is wITTM$_{\omega,\omega}$-decidable.*

Proof We note that every relation on natural numbers is wITTM-decidable if and only if it is wITTM-writable. Similarly, every relation on natural numbers is wITTM$_{\omega,\omega}$-decidable if and only if it is wITTM$_{\omega,\omega}$-writable. Therefore, the result follows from Theorem 27. □

From Theorem 29, we infer the following result.

Theorem 30 *For every function $f : \omega \to \omega$, f is wITTM-total-computable if and only if it is wITTM$_{\omega,\omega}$-total-computable.*

Using the halting problem for wITTMs we can show that some sets are strictly semi-decidable for wITTMs and similarly for wITTM$_{\omega,\omega}$s. Let h^- be the halting set for wITTMs, i.e. the set of all programs $e \in \omega$ such that, when implemented on a wITTM M with input e, M concludes its computation. Let $h^-_{\omega,\omega}$ be the halting set for wITTM$_{\omega,\omega}$s.

Proposition 31 *h^- is wITTM-semi-decidable but not wITTM-decidable.*

Proposition 32 *$h^-_{\omega,\omega}$ is wITTM$_{\omega,\omega}$-semi-decidable but not wITTM$_{\omega,\omega}$-decidable.*

The classic proof of the undecidability of the halting set for Turing machines works in these cases too. Suppose that h^- is wITTM-decidable. Then define a program p that checks whether e converges on input e. Define p such that, if e converges on input e, then p diverges. Moreover, define p such that, if e diverges on input e, then p converges. So, p on input p both diverges and converges, which is absurd. The same holds for $h^-_{\omega,\omega}$ and wITTM$_{\omega,\omega}$. From Propositions 31 and 32, one derives the following statement.

Corollary 33 *Some sets are wITTM-semi-decidable but not wITTM-decidable. Similarly, some sets are wITTM$_{\omega,\omega}$-semi-decidable but not wITTM$_{\omega,\omega}$-decidable.*

Since ω^2 is the set of all halting times of wITTMs (Halting Time Theorem 5), an ITTM can simulate the behaviour of an wITTM and notice whether it halts or not on a given input. Therefore, we can state the following proposition.

Proposition 34 *h^- is ITTM-computable.*

Proof Define an ITTM M that, for every wITTM N and every $f \in 2^\omega$, understands whether the computation of N on input r converges or not. Note that, if N diverges, then it either hangs or it loops forever. Define M such that it both carries out the computation of N and keeps track of

1. whether the content of a cell used to mimic N's computation changes infinitely often
2. whether the content of every cell used to mimic N's computation never changes between two successive limit steps.

In the first case, M discovers that N's computation hangs. In the second case, M determines that N loops forever. In both cases, M realizes that N's computation diverges. In this way, mimicking a wITTM implementing program e on input e, M

understands whether e is in h^- or not. The details are the following. Without modifying its computational power, we assume that M has five scratch tapes: $s_0, \ldots, s_4$. We decide that, on its input tape (i), first scratch tape (s_0), and output tape (o), M does exactly what N does on its input tape, scratch tape, and output tape. For every $n \in \omega$,

1. We use $s_1(n)$ to check whether the content in $i(n)$ changes infinitely often. Every time that the content of $i(n)$ changes, M changes the content of $s_1(n)$ to 1 and then to 0. At the next limit step, $s_1(n) = 1$ if and only if the content of $i(n)$ has changed infinitely often. If this is the case, M knows that N has hanged. Therefore M says that $e \notin h^-$.
2. We use $s_2(n)$ to check whether the content in $s_0(n)$ changes infinitely often and $s_3(n)$ to check whether the content in $o(n)$ changes infinitely often. The procedures are analogous to the one described above.
3. We use $s_4(0)$ to keep track of whether any cell on either i, s_0, or o changes at least once between two successive limit steps (or between the initial step and step ω). The first time that the content of a cell on either i, s_0, or o changes between step 0 and step ω, M put $s_4(0) = 1$. After step ω, M checks whether $s_4(0) = 0$. By Lemma 1, $s_4(0) = 0$ if and only if N will loop forever. Therefore, if $s_4(0) = 0$, then M knows that N's computation diverges and $e \notin h^-$. □

It is useful to note that one cannot use the idea in the proof of Proposition 34 to show that a wITTM$_{\omega,\omega}$ can compute the halting set of wITTMs. As shown in Propositions 4 and 6, a wITTM can hang or start looping exactly at step ω^2. Therefore, any wITTM$_{\omega,\omega}$ that simulates the behavior of a wITTM will also arrive at step ω^2, and its computation will diverge.

Now we briefly describe how an ITTM can decide the halting set for wITTM$_{\omega,\omega}$s. A simple way to do that is to observe that infinite time Turing machines with a bidimensional input tape and a bidimensional scratch tape (ITTM$_{\omega,\omega}$s) are as powerful as regular ITTMs. Then one can adapt the proof of Proposition 34 to show that an ITTM with bidimensional tapes (and, given the equivalence, a regular ITTM) can compute the halting set of wITTM$_{\omega,\omega}$.

We show that ITTMs and ITTM$_{\omega,\omega}$s are computationally equivalent. To do so, we define variants of ITTMs analogous to those defined for wITTMs. We define infinite time Turing machines with finitely many extra input and scratch tapes and one head (ITTM$_{<\omega,<\omega}$s) and infinite time Turing machines with finitely many extra input and scratch tapes and corresponding heads (ITTM$_{<\omega}$s). One can readily adapt the proofs of Equivalence Theorem 15 and Equivalence Theorem 20 to show that ITTM $\equiv$ ITTM$_{<\omega,<\omega}$ and ITTM$_{<\omega}$ $\equiv$ ITTM$_{\omega,\omega}$. The following proposition shows that ITTM$_{<\omega,<\omega}$ $\equiv$ ITTM$_{<\omega}$.

Proposition 35 *ITTMs and ITTM$_{<\omega}$s are computationally equivalent.*

Proof As noted above, ITTM $\equiv$ ITTM$_{<\omega,<\omega}$ and ITTM$_{<\omega}$ $\equiv$ ITTM$_{\omega,\omega}$. Moreover, one can readily observe that ITTM$_{<\omega,<\omega}$ $\leq$ ITTM$_{<\omega}$. Therefore, it remains to show that ITTM$_{<\omega}$ $\leq$ ITTM$_{<\omega,<\omega}$. We show that, for every $n \in \omega$, for some wITTM$_{<\omega,<\omega}$ M, for every wITTM$_{<\omega}$ N, for every sequence $r \in (2^\omega)^{n+1}$,

1. if $N(r)\downarrow$, then $M(r) = N(r)$ and

2. if $N(r)\uparrow$, then $M(r)\uparrow$.

We take M to have $n+1$ input tapes and $2(n+1)$ scratch tapes. Let us refer to the input tapes of M by $i_0, \dots, i_n$, the scratch tapes of M by $s_0, \dots, s_{2n+1}$, and the output tape of M by o. Let us refer to the input tapes of N by $I_0, \dots, I_n$, the scratch tapes of N by $S_0, \dots, S_n$, and the output tape of N by O. We use the single head of M to sequentially simulate the operations that the n many different heads of N carries out simultaneously at a single step. We use i_k and s_{2k} to replicate the content of I_k and S_k, respectively. We use s_{2k+1} to keep track of which position was visited last on the tapes i_k and s_k. Suppose, for example, that N has exactly two heads A and B. M first carries out on $(i_0(k), s_0(k), o(k))$ what A does on $(I_A(k), S_A(k), O(K))$. Then M moves left or right according to what N does at the corresponding step in its computation. Then M writes 1 on $s_1(k)$ and replaces the 1 that it had written previously on s_1 by 0. Then M looks for the unique cell containing 1 on s_3. Suppose that $s_3(k') = 1$. This is the signal that M needs to carry out on $(i_2(k'), s_2(k'), o(k'))$ what, at the corresponding step in its computation, N carries out on $(I_B(k'), S_B(k'), O_(k'))$. Then M sets $s_3(k') = 0$. Then, M prints 1 on $s_3(k'+1)$, if N moves right, and M prints 1 on $s_3(k'-1)$ otherwise. Then M looks for the unique 1 on s_1 and repeats the same procedure.

It remains to explain how M can understand whether two different heads of N try to write two different symbols on the same cell of N's output tape at the same time. One way is to introduce two new tape symbols X and Y. We then require M to print X on its output tape if N prints 0 and Y if N prints 1. If N tries to write two different symbols on the same cell at the same time, M will be in the situation in which it tries either to write X on a cell containing Y or Y on a cell containing X. In this case, we stipulate that M prints alternatively 0 and 1 on the same cell forever. Thus, the computation of M will diverge. If this situation does not arise, before mimicking the next step in N's computation, M scans the output tape and replaces every X with 0 and every Y with 1. □

We observe that the idea in the proof of Proposition 35 cannot be used to prove that wITTMs are computationally equivalent to wITTM$_{\omega,\omega}$s via showing that wITTM$_{<\omega,<\omega}$s are computationally equivalent to wITTM$_{<\omega}$s. The reason is that there is no guarantee that every cell in every scratch tape used to keep track of the positions of the various heads of a wITTM$_{<\omega}$s changes its value only finitely often.

From Proposition 35 and adapting the proof of Corollary 33, we can then prove the following proposition.

Proposition 36 *$h^-_{\omega,\omega}$ is ITTM-computable.*

5 Conclusion

We studied infinite time Turing machines where we replaced Hamkins and Lewis' *Lim Sup* rule with the rule *Eventually Constant*. Insofar as the notion of infinitary computation is legitimate, wITTMs are a natural extension of classical computability to infinite time computation. We showed that one can carry out real arithmetic with these machines and much more, like deciding all the arithmetic sets. We studied hardware variants of these machines showing that they are all strictly less powerful than regular infinite time Turing machines. We showed that all these weak models of infinitary computations agree on the computable functions on natural numbers, i.e. the functions on natural numbers that are computable according to each model of computability are the same. However, the question of whether these models are all computationally equivalent is still open.

Acknowledgements This work is largely a part of my master thesis [2] under the supervision of Julia F. Knight. I am very grateful to Julia F. Knight for her skillful guidance and insightful comments at every stage of the project. I would like to warmly thank Quinn Culver for numerous, extremely helpful discussions. I am also grateful to Merlin Carl, Dan Turetsky, and Greg Igusa for useful discussions and to an anonymous reviewer for valuable suggestions. I am also grateful to everyone that attended to presentations of earlier versions of this work either at the Notre Dame Computability seminar or at the CiE 2015 meeting.

References

1. Beeson, M.: Constructive geometry and the parallel postulate. Bull. Symb. Log. **22**(1), 1–104 (2016)
2. Bianchetti, M.: Infinite time computation: strong and weak infinite time turing machines. Master's thesis, University of Notre Dame (2017)
3. Carl, M.: Ordinal computability. An introduction to infinitary machines (forthcoming)
4. Descartes, R.: Discours de la methode pour bien conduire sa raison, & chercher la verité dans les sciences: plus la doptrique, les meteores, et la geometrie, qui sont des essais de cete methode. Maire, Leiden (1637)
5. Hamkins, J.D., Lewis, A.: Infinite time turing machines. J. Symb. Log. **65**(2), 567–604 (2000)
6. Rogers, H.: Theory fo Recursive Functions and Effective Computability. McGraw-Hill, New York (1967)