An Enhanced Theory of Infinite Time Register Machines*

Peter Koepke[1] and Russell Miller[2]

[1] Mathematisches Institut, Universität Bonn, Germany, koepke@math.uni-bonn.de
[2] Queens College and The Graduate Center, City University of New York, USA, Russell.Miller@qc.cuny.edu

Abstract. *Infinite time register machines* (ITRMs) are register machines which act on natural numbers and which are allowed to run for arbitrarily many ordinal steps. Successor steps are determined by standard register machine commands. At limit times a register content is defined as a lim inf of previous register contents, if that limit is finite; otherwise the register is *reset* to 0. (A previous weaker version of infinitary register machines, in [6], would halt without a result in case of such an overflow.) The theory of infinite time register machines has similarities to the infinite time Turing machines (ITTMs) of Hamkins and Lewis. Indeed ITRMs can decide all Π^1_1 sets, yet they are strictly weaker than ITTMs.

1 Introduction

Joel D. Hamkins and Andy Lewis [3] defined *infinite time* Turing *machines* (ITTMs) by letting an ordinary Turing machine run for arbitrarily many ordinal steps, taking appropriate limits at limit times. An ITTM can compute considerably more functions than a standard Turing machine. In this paper we introduce *infinite time register machines* (ITRMs) which may be seen as ordinary *register* machines running for arbitrarily many ordinal steps. Successor steps are determined by standard register machine commands. At limit times the register contents are defined as lim inf's of the previous register contents, if that limit is finite; otherwise the register is *reset* to 0.

Our ITRMs may be viewed as a specialization of the *ordinal register machines* (ORMs) examined in [8]. The stages are still allowed to range over all ordinals, but we now have a space bound of ω on the contents of the registers. Of course, this requires a rule for the action of the machine when a register overflows. In previous versions (see for example [6]), the machines halted or crashed when encountering an overflow; those machines exactly corresponded to hyperarithmetic definitions. Our machines reset a register to 0 whenever it overflows. We view this as a more natural rule, defining richer descriptive classes which are more in analogy with the ITTM-definable classes, and we believe that the theorems

* Keywords: ordinal computability, hypercomputation, infinitary computation, register machine.

in this paper support our view. Therefore, we propose to use the name *infinite time register machine* to refer to our machines in this paper. The machines defined in [6] by the first author could be called *non-resetting infinite time register machines*.

Our ITRMs are to ORMs as the ITTMs of Hamkins and Lewis are to *ordinal Turing machines*, or OTMs, as defined in [7]. In both cases the ordinal machines have unbounded time and space, whereas in the operation of traditional finite time machines, both time and space were bounded by ω. ITTMs and ITRMs fall in between, with ω-much space but unbounded time, hence are denoted as "infinite time" machines. With the bound on space, of course, the ITRMs necessarily follow different procedures than the ORMs at limit stages. (For ITTMs and OTMs the corresponding difference concerns head location, not cell contents.)

Many results for ITRMs in this paper reflect this connection with ITTMs. Notably, we show that ITRMs are Π^1_1-complete in the sense that for any lightface Π^1_1-set A of reals there is an ITRM such that a given real x is accepted by the machine iff $x \in A$ (Theorem 2). In particular the class WO of codes for wellorders is ITRM-decidable (Theorem 1), and likewise ITTM-decidable (by results in [3]). On the other hand ITRMs are strictly weaker than ITTMs because the latter are able to solve the halting problem for the former (Theorem 3). Moreover, for a given number N of registers, the halting problem for ITRMs with N registers is ITRM-decidable, using of course more registers (Theorem 4). In further research we plan to develop the theory of ITRMs along the lines of the ITTMs in [3].

2 Infinite time register machines

We base our presentation of infinite time machines on the *unlimited register machines* of [1].

Definition 1. *An* unlimited register machine *URM has registers* $R_0, R_1, \ldots$ *which can hold* natural numbers. *A register program consists of commands to increase or to reset a register. The program may jump on condition of equality of two registers.*

An URM program *is a finite list* $P = I_0, I_1, \ldots, I_{s-1}$ *of* instructions, *each of which may be of one of five kinds:*

a) *the* zero instruction $Z(n)$ *changes the contents of* R_n *to* 0, *leaving all other registers unaltered;*
b) *the* successor instruction $S(n)$ *increases the natural number contained in* R_n *by* 1, *leaving all other registers unaltered;*
c) *the* oracle *instruction* $O(n)$ *replaces the content of the register* R_n *by the number* 1 *if the content is an element of the oracle, and by* 0 *otherwise;*
d) *the* transfer instruction $T(m, n)$ *replaces the contents of* R_n *by the natural number contained in* R_m, *leaving all other registers unaltered;*
e) *the* jump instruction $J(m, n, q)$ *is carried out within the program* P *as follows: the contents* r_m *and* r_n *of the registers* R_m *and* R_n *are compared, all registers*

are left unaltered; then, if $r_m = r_n$, the URM proceeds to the qth instruction of P; if $r_m \neq r_n$, the URM proceeds to the next instruction in P.

Since the program is finite, it can use only finitely many of the registers, and the exact number of registers used will often be important. The instructions of the program can be addressed by their indices which are called program states. *At each ordinal time τ the machine will be in a* configuration *consisting of a program state $I(\tau) \in \omega$ and the register contents which can be viewed as a function $R(\tau) : \omega \to \omega$. $R(\tau)(n)$ is the content of the register R_n at time τ. We also write $R_n(\tau)$ instead of $R(\tau)(n)$.*

Definition 2. *Let $P = I_0, I_1, \ldots, I_{s-1}$ be an* URM *program. Let $Z \subseteq \omega$, which will serve as an oracle. A pair*

$$I : \theta \to \omega, R : \theta \to ({}^{\omega}\omega)$$

is an (infinite time register) computation *by P if the following hold:*

a) θ *is an ordinal or $\theta =$* Ord*;* θ *is the* length *of the computation;*
b) $I(0) = 0$*; the machine starts in state* 0*;*
c) If $\tau < \theta$ and $I(\tau) \notin s = \{0, 1, \ldots, s-1\}$ then $\theta = \tau + 1$; the machine halts *if the machine state is not a program state of P;*
d) If $\tau < \theta$ and $I(\tau) \in s$ then $\tau + 1 < \theta$; the next configuration is determined by the instruction $I_{I(\tau)}$, with $I(\tau + 1) = I(\tau) + 1$ unless otherwise specified:
  i. if $I_{I(\tau)}$ is the zero instruction $Z(n)$ then define $R(\tau + 1) : \omega \to$ Ord *by setting $R_k(\tau + 1)$ to be 0 (if $k = n$) or $R_k(\tau)$ (if not).*
  ii. if $I_{I(\tau)}$ is the successor instruction $S(n)$ then define $R_k(\tau + 1)$ to be $R_k(\tau) + 1$ (if $k = n$) or $R_k(\tau)$ (if not).
  iii. if $I_{I(\tau)}$ is the oracle instruction $O(n)$ then define $R_k(\tau+1)$ to be $R_k(\tau)$ (if $k \neq n$); or 1 (if $k = n$ and $R_k(\tau) \in Z$); or 0 (if $k = n$ and $R_k(\tau) \notin Z$).
  iv. if $I_{I(\tau)}$ is the transfer instruction $T(m, n)$ then define $R_k(\tau + 1)$ to be $R_m(\tau)$ (if $k = n$) or $R_k(\tau)$ (if not).
  v. if $I_{I(\tau)}$ is the jump instruction $J(m, n, q)$ then let $R(\tau + 1) = R(\tau)$, and set $I(\tau + 1) = q$ (if $R_m(\tau) = R_n(\tau)$) or $I(\tau + 1) = I(\tau) + 1$ (if not).
e) If $\tau < \theta$ is a limit ordinal, then $I(\tau) = \liminf_{\sigma \to \tau} I(\sigma)$ and

$$\forall k \in \omega \; R_k(\tau) = \begin{cases} \liminf_{\sigma \to \tau} R_k(\sigma), & \text{if } \liminf_{\sigma \to \tau} R_k(\sigma) < \omega \\ 0, & \text{if } \liminf_{\sigma \to \tau} R_k(\sigma) = \omega. \end{cases}$$

By the second clause in the definition of $R_k(\tau)$ the register is reset *in case $\liminf_{\sigma \to \tau} R_k(\sigma) = \omega$.*

The computation is obviously determined recursively by the initial register contents $R(0)$, the oracle Z and the program P. We call it the (infinite time register) computation by P with input $R(0)$ *and oracle Z. If the computation halts then $\theta = \beta + 1$ is a* successor *ordinal and $R(\beta)$ is the final register content. In this case we say that P* computes *$R(\beta)(0)$ from $R(0)$ and the oracle Z, and we write $P : R(0), Z \mapsto R(\beta)(0)$.*

Definition 3. *An n-ary partial function $F : \omega^n \rightharpoonup \omega$ is* computable *if there is a register program P such that for every n-tuple $(a_0, \ldots, a_{n-1}) \in \mathrm{dom}(F)$ holds*

$$P : (a_0, \ldots, a_{n-1}, 0, 0, \ldots), \emptyset \mapsto F(a_0, \ldots, a_{n-1}).$$

Here the oracle instruction is not needed.

Obviously any standard recursive function is computable.

Definition 4. *A subset $x \subseteq \omega$, i.e., a (single) real number, is* computable *if its characteristic function χ_x is computable.*

A subset $A \subseteq \mathcal{P}(\omega)$ is computable *if there is a register program P, and an oracle $Y \subseteq \omega$ such that for all $Z \subseteq \omega$:*

$$Z \in A \text{ iff } P : (0, 0, \ldots), Y \times Z \mapsto 1, \text{ and } Z \notin A \text{ iff } P : (0, 0, \ldots), Y \times Z \mapsto 0$$

where $Y \times Z$ is the cartesian product of Y and Z with respect to the pairing function

$$(y, z) \mapsto \frac{(y + z)(y + z + 1)}{2} + z.$$

Here we allow a single real parameter Y (or equivalently, finitely many such parameters), mirroring the approach in [6].

3 Computing $\boldsymbol{\Pi^1_1}$-sets

We describe an ITRM-program to check the oracle Z for illfoundedness. Illfoundedness will be witnessed by some infinite descending chain. Initial segments of such a chain will be kept on a finite *stack* of natural numbers. Code a stack $(r_0, \ldots, r_{m-1})$ by $r = 2^m \cdot 3^{r_0} \cdot 5^{r_1} \cdots p_m^{r_{m-1}}$ where p_i is the i-th prime number. In the subsequent program we shall treat one register as a stack, with content `stack` equal to r above, and with associated operations `push`, `pop`, `length-stack`, `stack-is-decreasing`; this last predicate checks that the elements of the stack, except possibly the bottom element, form a decreasing sequence in the oracle Z. All of these are computable by an ITRM. The specific coding of stack contents leads to a controlled limit behaviour:

Proposition 1. *Let $\alpha < \tau$ where τ is a limit ordinal. Assume that in some ITRM-computation using a stack, the stack contains $r = (r_0, \ldots, r_{m-1})$ for cofinally many times below τ and that all contents in the time interval (α, τ) are endextensions of $r = (r_0, \ldots, r_{m-1})$. Then at time τ the stack contents are $r = (r_0, \ldots, r_{m-1})$.*

The following program P on an ITRM outputs `yes`/`no` depending on whether the oracle Z codes a wellfounded relation. The program is a backtracking algorithm which searches for a "leftmost" infinite descending chain in Z. A stack is used to organize the backtracking. We present the program in simple pseudo-code and assume that it is translated into a register program according to Definition

1 so that the order of commands is kept. Also the stack commands like `push` are understood as *macros* which are inserted into the code with appropriate renaming of variables and statement numbers. The ensuing Lemma explains the operation of the program and proves its correctness.

```
      push 1; %% marker to make stack non-empty
      push 0; %% try 0 as first element of descending sequence
      FLAG=1; %% flag that fresh element is put on stack
Loop: Case1: if FLAG=0 and stack=0 %% inf descending seq found
          then begin; output 'no'; stop; end;
      Case2: if FLAG=0 and stack=1 %% inf descending seq not found
          then begin; output 'yes'; stop; end;
      Case3: if FLAG=0 and length-stack > 1
      %% top element cannot be continued infinitely descendingly
          then begin; %% try next
          pop N;
          push N+1;
          FLAG:=1; %% flag that fresh element is put on stack
          goto Loop;
          end;
      Case4: if FLAG=1 and stack-is-decreasing
          then begin;
          push 0; %% try to continue sequence with 0
          FLAG:=0; FLAG:=1; %% flash the flag
          goto Loop;
          end;
      Case5: if FLAG=1 and not stack-is-decreasing
          then begin;
          pop N;
          push N+1; %% try next
          FLAG:=0; FLAG:=1; %% flash the flag
          goto Loop;
          end;
```

Notice that the program will always loop back to `Loop` until it halts.

Lemma 1. *Let $I : \theta \to \omega, R : \theta \to ({}^{\omega}\omega)$ be the computation by P with oracle Z and trivial input $(0, 0, \ldots)$. Then the computation satisfies:*

a) Suppose the machine is in state `Loop` *with stack contents $(1, r_0, \ldots, r_{m-1})$ so that $(r_0, \ldots, r_{m-1})$ descend strictly in Z. Moreover suppose that* `Flag=1` *and that Z is wellfounded below r_{m-1}. Then the machine will reach the state* `Loop` *with the same stack contents and* `Flag=0` *after a certain interval of time; during that interval, $(1, r_0, \ldots, r_{m-1})$ will always be an initial segment of the stack.*

b) Suppose the machine is in state `Loop` *with stack contents $(1, r_0, \ldots, r_{m-1})$ so that $(r_0, \ldots, r_{m-1})$ descend strictly in Z. Moreover suppose that* `Flag=1`

and that Z is illfounded below r_{m-1}. Let r_m be the smallest integer such that $r_m Z r_{m-1}$ and Z is illfounded below r_m . Then the machine will reach the state `Loop` *with stack contents* $(1, r_0, \ldots, r_{m-1}, r_m)$ *and* `Flag=1` *after a certain interval of time; during that interval,* $(1, r_0, \ldots, r_{m-1})$ *will always be an initial segment of the stack.*

c) *If Z is wellfounded then the computation stops with output* `'yes'`.

d) *If Z is illfounded then the computation stops with output* `'no'`.

Proof. a) is proved by induction on r_{m-1} in the wellfounded part of Z. So consider a situation $(1, r_0, \ldots, r_{m-1})$ as in a) and assume that a) already holds for all appropriate stacks $(1, r'_0, \ldots, r'_{m'-1})$ with $r'_{m'-1} Z r_{m-1}$. By `Case4`, `Case3`, and the inductive assumption, the machine will check through all extensions $(1, r_0, \ldots, r_{m-1}, N)$ with $N \in \omega$ of the stack and always get to state `Loop` with `Flag=0`. The limit of these checks is a stack $(1, r_0, \ldots, r_{m-1})$ with `Flag=0`, as required.

b) Consider the situation described in b). The program checks through all extensions $(1, r_0, \ldots, r_{m-1}, N)$ with $N < r_m$ of the stack. `Case5` rejects those N which fail $N Z r_{m-1}$, and part (a) shows that the others are also rejected. So `Case3` finally puts r_m on the stack, with `Flag=1`.

c) and d) follow from a) and b) resp.

Parts c) and d) of the Lemma imply immediately:

Theorem 1. *The set* $\mathrm{WO} = \{Z \subseteq \omega \mid Z$ *codes a wellorder*$\}$ *is computable by an ITRM.*

Theorem 2. *Every Π^1_1 set $A \subseteq \mathcal{P}(\omega)$ is ITRM-computable.*

Proof. Let f be a recursive function so that $Y \in A \leftrightarrow f(Y) \in \mathrm{WO}$. Given a real Y an ITRM can decide whether $Y \in A$ by letting the above WO-algorithm run on $f(Y)$. Note that the algorithm needs to decide whether certain integers stand in the relation $f(Y)$. This can be reduced to computing a certain digit of $f(Y)$ which is possible using the oracle Y and a fixed algorithm for computing f.

Since Π^1_1-sets can be decided, it is also possible to decide Boolean combinations of Π^1_1-sets by ITRMs. By induction on ordertypes one can prove a running time estimate for the WO-algorithm:

Lemma 2. *For an oracle Z coding a well order of ordertype α the* WO*-program runs at least α steps before it halts.*

4 ITRMs, ITTMs, and halting problems

A computation by an ITRM can be simulated by an ITTM. If the register R_m contains the number i this can be represented as an initial segment of i 1's on the m-th tape of an ITTM. If λ is a limit ordinal and the contents of the register R_m yield $\liminf_{\tau \to \lambda} R_m(\tau) = i^* \leqslant \omega$ then the m-th tape will hold an initial

segment of i^* 1's at time λ. If i^* is finite, this is the correct simulation of the ITRM. If $i^* = \omega$ this may be checked by an auxiliary program which then *resets* the register to 0. Thus every class of reals which is computable by an ITRM is computable by an ITTM, and hence must be Δ^1_2, by Theorem 2.5 in [3].

In fact ITRMs are strictly weaker than ITTMs. A *configuration* is a tuple (I, R) of a program state I and register contents $R : \omega \to \omega$ where $R(n) = 0$ for almost all $n < \omega$. The following halting criterion for ITRMs uses a wellfounded pointwise partial order of configurations:

$$(I_0, R_0) \leqslant (I_1, R_1) \text{ iff } I_0 \leqslant I_1 \text{ and } \forall n < \omega \; R_0(n) \leqslant R_1(n).$$

Lemma 3. *Let*

$$I : \theta \to \omega, R : \theta \to ({}^{\omega}\omega)$$

be the infinite time register computation by P with input $(0, 0, \ldots)$ and oracle Z. Then this computation does not *halt iff there are $\tau_0 < \tau_1 < \theta$ such that $(I(\tau_0), R(\tau_0)) = (I(\tau_1), R(\tau_1))$ and*

$$\forall \tau \in [\tau_0, \tau_1] \; (I(\tau_0), R(\tau_0)) \leqslant (I(\tau), R(\tau)).$$

Proof. Assume that the computation does not halt. Let A be the set of all configurations which occur class-many times in this computation, and fix a stage τ^- after which only configurations in A occur. We claim that A is downwards directed in the partial order of configurations: for $(I_0, R_0), (I_1, R_1) \in A$ choose an ascending ω-sequence $\tau^- < \tau_0 < \tau_1 < \cdots$ of stages such that each (I_i, R_i) occurs at all stages of the form $\tau_{2\cdot k+i}$ with $i < 2$. Then the configuration (I, R) occuring at stage $\tau = \sup_n \tau_n$ has $(I, R) \leqslant (I_0, R_0)$ and $(I, R) \leqslant (I_1, R_1)$, by the rules for limit stages.

Let (I_0, R_0) be the unique $\leqslant$-minimal element of A. Choose stages τ_0, τ_1 such that $t^- < t_0 < t_1 < \theta$ and $(I(\tau_0), R(\tau_0)) = (I(\tau_1), R(\tau_1)) = (I_0, R_0)$. This is the situation required by the lemma.

For the converse assume that there are $\tau_0 < \tau_1 < \theta$ such that $(I(\tau_0), R(\tau_0)) = (I(\tau_1), R(\tau_1))$ and

$$\forall \tau \in [\tau_0, \tau_1] \; (I(\tau_0), R(\tau_0)) \leqslant (I(\tau), R(\tau)).$$

Then one can easily show by induction, using the lim inf rules:
If $\sigma \geqslant \tau_0$ is of the form $\sigma = \tau_0 + (\tau_1 - \tau_0) \cdot \alpha + \beta$ with $\beta < \tau_1 - \tau_0$ then

$$(I(\sigma), R(\sigma)) = (I(\tau_0 + \beta), R(\tau_0 + \beta)).$$

In particular the computation will not stop.

Theorem 3. *The* halting problem *for ITRMs*

$$\{(P, Z) \mid P \textit{ is a register program, } Z \subseteq \omega, \textit{ and the computation by } P \textit{ with input } (0, 0, \ldots) \textit{ and oracle } Z \textit{ halts}\}$$

is decidable by an ITTM with oracle Z.

Proof. The criterion of Lemma 3 can be implemented on an ITTM with an auxiliary tape on which we have one cell for each possible configuration of the ITRM. We use the ITTM to simulate the ITRM computation by a program P with input $(0, 0, \ldots)$ and oracle Z. At stage τ of the simulation we erase from the auxiliary tape all 1's for configurations which are not $\leqslant (I(\tau), R(\tau))$, and put a 1 in the cell for the configuration $(I(\tau), R(\tau))$. If there was already a 1 in this cell, then we conclude from Lemma 3 that the computation never halts. At limit stages the same procedure applies. (There may be infinitely many 1's on the auxiliary tape at a limit stage, of which cofinitely many will immediately be erased. For an ITTM, this poses no difficulty.) These two processes continue until either the simulated ITRM computation halts or we conclude as above that it will never halt. By Lemma 3, one of these alternatives must happen.

For a fixed number of registers these ideas can be transfered to an ITRM (with more registers).

Theorem 4. *The restricted halting problem*

$$\{(P, Z) \mid P \text{ is a register program using at most } N \text{ registers, } Z \subseteq \omega, \text{ and the computation by } P \text{ with input } (0, 0, \ldots) \text{ and oracle } Z \text{ halts}\}$$

is decidable by an ITRM with oracle Z, for every $N < \omega$.

Proof. We introduce some notation to handle configurations of the N register machine. View a configuration (I, R) as the $(N + 1)$-sequence

$$(R(0), \ldots, R(N - 1), I)$$

and use letters $c, c', \ldots$ to denote configurations. Write $c \leqslant c'$ iff $\forall m \leqslant N\ c(m) \leqslant c'(m)$. Let $I : \theta \to \omega$, $R : \theta \to ({}^{\omega}\omega)$ be the infinite time resetting register computation by P with input $(0, 0, \ldots)$ and oracle Z. The computation is a sequence $(c(\tau) | \tau < \theta)$ of configurations.

By Lemma 3, the computation does not stop ($\theta = \infty$) iff

$$\exists \sigma < \tau < \theta\ (c(\sigma) = c(\tau) \wedge \forall \sigma' \in [\sigma, \tau]\ c(\sigma) \leqslant c(\sigma')).$$

This motivates the definition

$$C(\tau) = \{c(\sigma) | \sigma < \tau \wedge \forall \sigma' \in [\sigma, \tau]\ c(\sigma) \leqslant c(\sigma')\}.$$

Then the halting criterion is simply

$$\exists \tau (c(\tau) \in C(\tau)).$$

Note that the initial configuration $(0, \ldots, 0)$ is an element of $C(\tau)$ for all $\tau > 0$.

The Theorem will be proved by showing that (a code for) $C(\tau)$ can be easily computed, and indeed by an ITRM. For technical reasons we introduce some auxiliary sequences of configuration sets. For $m \leqslant N$ let $C_m(\tau)$ be the *finite* set

$$C_m(\tau) = \{c(\sigma) | \sigma < \tau \wedge \forall \sigma' \in [\sigma, \tau]\ c(\sigma) \leqslant c(\sigma') \wedge \forall i \leqslant N\ c(\sigma)(i) \leqslant c(\tau)(m)\}.$$

Obviously $C(\tau) = C_{m_0}(\tau)$ where $c(\tau)(m_0) = \max_{i \leqslant N} c(\tau)(i)$. To handle sets of the form $C_m(\tau)$ as natural numbers and register contents we assume that we have a recursive enumeration or Gödelization $c_0, c_1, \ldots$ of configurations with N registers. Finite sets C of configurations can be coded by the natural number

$$C^* = \prod_{c_k \in C} p_k \ ,$$

which can be stored in a machine register.

Consider a simulation of the computation $(c(\tau)|\tau < \theta)$ on some register machine with sufficiently many registers. We argue that the sequence $(C(\tau)^*|\tau < \theta)$ can be uniformly computed alongside the simulation, which solves the halting problem. We proceed by induction on $\tau < \theta$.

$C(0) = \{(0, \ldots, 0)\}$ only contains the initial configuration.

If $C(\tau)$, $c(\tau)$ and $c(\tau + 1)$ are given, then

$$C(\tau+1) = \begin{cases} \{c \in C(\tau) | c \leqslant c(\tau+1)\} \cup \{c(\tau)\}, \text{ if } c(\tau) \leqslant c(\tau+1); \\ \{c \in C(\tau) | c \leqslant c(\tau+1)\}, \text{ else.} \end{cases}$$

Hence $C(\tau+1)^*$ can be computed by an ordinary register machine from $C(\tau)^*$, $c(\tau)$, and $c(\tau+1)$.

Finally consider the limit time $\lambda < \theta$.

In case that $c(\lambda) = (0, \ldots, 0)$ then $C(\lambda) = \{(0, \ldots, 0)\}$. $C(\lambda)^*$ is easily computable, and moreover the criterion for divergence of the computation is fulfilled. So consider the case that $c(\lambda) \neq (0, \ldots, 0)$. Choose m_0 such that

$$c(\lambda)(m_0) = \max_i c(\lambda)(i) > 0.$$

Then $C_{m_0}(\lambda) = C(\lambda)$ and
(1) $c(\lambda)(m_0) = \liminf_{\tau \to \lambda} c(\tau)(m_0)$.
(2) $\liminf_{\tau \to \lambda} C_{m_0}(\tau)^*$ exists and is finite.
Proof. By (1) there is a cofinal subset $T \subseteq \lambda$ such that

$$\forall \tau \in T \ \ c(\tau)(m_0) = c(\lambda)(m_0).$$

For $\tau \in T$ we have

$$C_{m_0}(\tau) \subseteq \{c \mid \forall i \leqslant N \ c(i) \leqslant c(\lambda)(m_0)\}.$$

The right-hand side is a fixed finite set. So for $\tau \in T$, $C_{m_0}(\tau)^*$ is bounded by some fixed integer. Thus the lim inf is finite. *qed*(2)
(3) Let $c_k \leqslant c(\lambda)$. Then $p_k | C_{m_0}(\lambda)^*$ iff $p_k | \liminf_{\tau \to \lambda} C_{m_0}(\tau)^*$.
Proof. Let $p_k | C_{m_0}(\lambda)^*$. Then $c_k \in C_{m_0}(\lambda)$. Take $\sigma < \lambda$ with $c_k = c(\sigma)$ such that for all $\sigma' \in [\sigma, \lambda]$, both $c_k \leqslant c(\sigma')$ and $c(\sigma')(m_0) \geq c(\lambda)(m_0)$. Then for all $\tau \in (\sigma, \lambda)$ we have $c_k \in C_{m_0}(\tau)$ and $p_k | C_{m_0}(\tau)^*$. Since $\liminf_{\tau \to \lambda} C_{m_0}(\tau)^*$ will be equal to one of those $C_{m_0}(\tau)^*$ we get that $p_k | \liminf_{\tau \to \lambda} C_{m_0}(\tau)^*$.

Conversely assume $p_k | \liminf_{\tau \to \lambda} C_{m_0}(\tau)^*$. Take $\tau_0 < \lambda$ such that $\forall \tau \in [\tau_0, \lambda]$ $c(\tau) \geqslant c(\lambda)$. Take $\tau_1 \in [\tau_0, \lambda)$ such that $\liminf_{\tau \to \lambda} C_{m_0}(\tau)^* = C_{m_0}(\tau_1)^*$. Then

$p_k|C_{m_0}(\tau_1)^*$, $c_k \in C_{m_0}(\tau_1)$ and by the choice of τ_0 also $c_k \in C_{m_0}(\tau)$ for all $\tau \in [\tau_1, \lambda)$. Since $c_k \leqslant c(\lambda)$ we have $c_k \in C_{m_0}(\lambda)$ and $p_k|C_{m_0}(\lambda)^*$. *qed*(3)

This means that $C_{m_0}(\lambda)^* = C(\lambda)^*$ can be computed from $(C_{m_0}(\tau)^*|\tau < \lambda)$ by a lim inf-operation. To compute the sequence $(C(\tau)^*|\tau < \theta)$ alongside $(c(\tau)|\tau < \lambda)$ we can use $N+1$ new registers $R_0, \ldots, R_N$ to store the values $C_0(\tau)^*, \ldots, C_N(\tau)^*$. Initially these registers are set to $\{(0, \ldots, 0)\}^*$. Given $C_0(\tau)^*, \ldots, C_N(\tau)^*$, $c(\tau)$, and $c(\tau{+}1)$ one can compute $C_0(\tau{+}1)^*, \ldots, C_N(\tau{+}1)^*$ by an ordinary register program on some extra registers and transfer these values to $R_0, \ldots, R_N$. For limit $\lambda < \theta$ the lim inf-rule sets $R_0, \ldots, R_N$ to

$$\liminf_{\tau \to \lambda} C_0(\tau)^*, \ldots, \liminf_{\tau \to \lambda} C_N(\tau)^*.$$

By (3), an ordinary register program on further extra registers can compute the value $C(\lambda)^* = C_{m_0}(\lambda)^*$, from which it can then compute

$$C_0(\lambda)^*, \ldots, C_N(\lambda)^*$$

and transfer them to $R_0, \ldots, R_N$.

This means for all $\tau \in [\omega, \theta)$, $C_m(\tau)$ will be the $(\tau+1)$-st value transferred to the register R_m, concluding the proof of Theorem 4.

The Theorem implies that the machines get eventually stronger by increasing the number of registers. As a consequence there cannot be a universal ITRM.

References

[1] N.J. Cutland. *Computability: An Introduction to Recursive Function Theory*. Perspectives in Mathematical Logic. Cambridge University Press (1980)

[2] I. Dimitriou, J.D. Hamkins, and P. Koepke, eds. *BIWOC – Bonn International Workshop on Ordinal Computability*, Bonn Logic Reports (2007)

[3] J.D. Hamkins and A. Lewis. Infinite Time Turing Machines. *J. Symbolic Logic*, 65(2), 567–604 (2000)

[4] J.D. Hamkins, D. Linetsky, and R. Miller. The complexity of quickly ORM-decidable sets. In S.B. Cooper et al. (eds.) *Computation and Logic in the Real World*, LNCS, vol. 4497, pp. 488–496. Springer, Heidelberg (2007)

[5] J.D. Hamkins and R. Miller. Post's problem for ordinal register machines. In S.B. Cooper et al. (eds.) *Computation and Logic in the Real World*, LNCS, vol. 4497, pp. 358–367. Springer, Heidelberg (2007)

[6] P. Koepke. Infinite time register machines. In A. Beckmann et al. (eds.) *Logical approaches to computational barriers*, LNCS, vol. 3988, pp. 257–266. Springer, Heidelberg (2006)

[7] P. Koepke. Turing computations on ordinals. *B. Symbolic Logic*, 11, 377-397 (2005)

[8] P. Koepke and R. Siders. Computing the recursive truth predicate on ordinal register machines. In Arnold Beckmann et al., editors, *Logical approaches to computational barriers*, *Computer Science Report Series* (7), 160–169 (2006)