Infinite Time Register Machines

Peter Koepke

University of Bonn
Mathematisches Institut
Beringstraße 1
D 53115 Bonn
Germany
Koepke@Math.Uni-Bonn.de

Abstract. *Infinite time register machines* (ITRMs) are register machines which act on natural numbers and which may run for arbitrarily many ordinal steps. Successor steps are determined by standard register machine commands, at limits the register contents are defined as lim inf's of the previous register contents. We prove that a real number is computable by an ITRM iff it is hyperarithmetic.

1 Introduction

In [2], Joel D. Hamkins and Andy Lewis define *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 analogy, we let a standard *register* machine run for arbitrarily many ordinal steps and call it an *infinite time register machines* (ITRM). An ITRM can carry out infinitely many steps of an ordinary register machine and can thus compute the halting problem. Indeed we show in Lemma 1 that it can compute any Δ_1^1 real number. Conversely it will be shown in Lemma 4 that if a computation by an ITRM halts then it halts before the Church-Kleene ordinal ω_1^{CK}. Hence all computable reals are in the admissible set $L_{\omega_1^{\mathrm{CK}}}$ (Lemma 5). Since the Δ_1^1-reals coincide with the reals in $L_{\omega_1^{\mathrm{CK}}}$ and with the *hyperarithmetic* reals (see [8]) this yields a new characterisation of the hyperarithmetic reals:

Theorem 1. *A real $x \subseteq \omega$ is computable by an infinite time register machine iff it is hyperarithmetic.*

This result was inspired by discussions with Joel Hamkins and Philip Welch at Oberwolfach in December 2005. Infinite time register machines belong to the following schema of machines which may all run for arbitrarily many ordinal steps. Let Ord be the class of ordinal numbers.

1.1: Infinite time Turing machines, ITTMs, with finitely many standard Turing tapes; every Σ_1^1 real and every Π_1^1 real is ITTM computable [2].

1.2: Ordinal Turing machines, OTMs, with finitely many Turing tapes of length ORD; a set of ordinals is OTM computable iff it is a constructible set of ordinals [3], [4], [6].

2.1: Infinite time register machines, ITRMs, as defined in this article; a real is ITRM computable iff it is hyperarithmetic (Δ^1_1).

2.2: Ordinal register machines, ORMs, with finitely many registers containing arbitrary ordinals; a set of ordinals is ORM computable iff it is a constructible set of ordinals [7].

2 Infinite Time Register Machines

We base our presentation of infinite time machines on the *unlimited register machines* as presented in [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 between 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 four 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 transfer instruction *$T(m,n)$ replaces the contents of R_n by the natural number contained in R_m, leaving all other registers unaltered;*
d) 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, but all the 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.*

The instructions of a register program can be addressed by their indices which are called program states*. At each ordinal time t the machine will be in a configuration consisting of a program state $I(t) \in \omega$ and the register contents which can be viewed as a function $R(t) : \omega \to \omega$. $R(t)(n)$ is the content of the register R_n at time t. We also write $R_n(t)$ instead of $R(t)(n)$.*

Definition 2. *Let $P = I_0, I_1, \ldots, I_{s-1}$ be an* URM *program. 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 $\theta = \mathrm{Ord}$; θ is the length *of the computation;*
b) $I(0) = 0$; the machine starts in state 0;

c) *If $t < \theta$ and $I(t) \notin s = \{0, 1, \ldots, s-1\}$ then $\theta = t + 1$; the machine* stops *if the machine state is not a program state of P;*

d) *If $t < \theta$ and $I(t) \in \mathrm{state}(P)$ then $t + 1 < \theta$; the next configuration is determined by the instruction $I_{I(t)}$:*

   i. *if $I_{I(t)}$ is the zero instruction $Z(n)$ then let $I(t+1) = I(t)+1$ and define $R(t+1) : \omega \to \mathrm{Ord}$ by*

$$R_k(t+1) = \begin{cases} 0, & \text{if } k = n \\ R_k(t), & \text{if } k \neq n \end{cases}$$

   ii. *if $I_{I(t)}$ is the successor instruction $S(n)$ then let $I(t+1) = I(t) + 1$ and define $R(t+1) : \omega \to \mathrm{Ord}$ by*

$$R_k(t+1) = \begin{cases} R_k(t) + 1, & \text{if } k = n \\ R_k(t), & \text{if } k \neq n \end{cases}$$

   iii. *if $I_{I(t)}$ is the transfer instruction $T(m, n)$ then let $I(t+1) = I(t)+1$ and define $R(t+1) : \omega \to \mathrm{Ord}$ by*

$$R_k(t+1) = \begin{cases} R_m(t), & \text{if } k = n \\ R_k(t), & \text{if } k \neq n \end{cases}$$

   iv. *if $I_{I(t)}$ is the jump instruction $J(m, n, q)$ then let $R(t+1) = R(t)$ and*

$$I(t+1) = \begin{cases} q, & \text{if } R_m(t) = R_n(t) \\ I(t) + 1, & \text{if } R_m(t) \neq R_n(t) \end{cases}$$

e) *If $t < \theta$ is a limit ordinal, the machine constellation at t is determined by taking inferior limits. If $\liminf_{r \to t} R_k(r) = \omega$ for some $k \in \omega$ then let $\theta = t$; the machine* stops *if one of the registers overruns; otherwise let*

$$\forall k \in \omega \;\; R_k(t) = \liminf_{r \to t} R_k(r);$$
$$I(t) = \liminf_{r \to t} I(r).$$

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

The definition of $I(t)$ for limit t can be motivated as follows. Since a program is finite its execution will lead to some (complex) looping structure involving loops, subloops and so forth. This can be presented by pseudo code like:

```
        ...
17:begin mainloop
           ...
```

```
     21:   begin subloop
                ...
     29:   end subloop
           ...
 32:end mainloop
        ...
```

Assume that for times $r \to t$ the main loop $(17-32)$ with its subloop $(21-29)$ is traversed cofinally often. Then at time t it is natural to put the machine at the start of the "main loop". Assuming that the lines of the program are enumerated in increasing order this corresponds to the lim inf rule

$$I(t) = \liminf_{r \to t} S(r).$$

The interpretation of programs yields associated notions of computability.

Definition 3. *An n-ary partial function $F : \omega^n \rightharpoonup \omega$ is* (ordinal register) 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*

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

Definition 4. *A subset $x \subseteq \omega$, i.e., a real number, is* (ordinal register) computable *if there is a register program P such that for every $m \in \omega$ holds*

$$P : (m, 0, 0, \ldots) \mapsto \chi_x(m),$$

where χ_x is the characteristic function of x.

Obviously any standard recursive function is ordinal register computable.

3 Computing Δ^1_1-Reals

For $e \in \omega$ let R_e denote the e-th recursively enumerable, binary relation on ω. If R_e is wellfounded, let $|R_e|$ denote the ordinal rank of R_e. Consider a *hyperarithmetic* real number x, i.e., $\{x\}$ is a parameter-free Δ^1_1-singleton. By standard representation theorems for Π^1_1-reals there exists a recursive function $f : \omega \to \omega$ such that for all $n \in \omega$:

$$n \in x \text{ iff } R_{f(n)} \text{ is a wellfounded relation.} \tag{1}$$

Since x is also Σ^1_1 the boundedness property for parameter-free Σ^1_1-sets implies the existence of an ordinal α less than the Church-Kleene ordinal ω_1^{CK} such that for all $n \in \omega$:

$$n \in x \text{ iff } R_{f(n)} \text{ is a wellfounded relation of rank } |R_{f(n)}| < \alpha. \tag{2}$$

The ordinal α is the ordertype of some recursive wellorder (ω, S). The right-hand side of (2) holds iff there is an orderpreserving embedding from $(\omega, R_{f(n)})$ into (ω, S).

More generally, consider any infinite time register computable relations (ω, R) and (ω, S) where (ω, S) is a wellorder. We shall define a register program P uniformly in programs for R and S which computes whether (ω, R) can be embedded orderpreservingly into (ω, S). This shows that the right-hand side of the equivalence (2) is infinite time register computable and proves

Lemma 1. *If $x \subseteq \omega$ is a hyperarithmetic real then x is computable by an infinite time register machine.*

For $r \in \omega$ let $\mathrm{TC}_R(r)$ be the transitive closure of r in R, i.e. the $\subseteq$-smallest set which contains r and is closed under R-predecessors. Define $\mathrm{TC}_S(s)$ similarly. Define a relation $r \sim s$ iff there is an orderpreserving map

$$\pi : (\mathrm{TC}_R(r), R) \to (\mathrm{TC}_S(s), S) \text{ with } \pi(r) = s.$$

If the relations R and S both have 0 as their maximum element, i.e.,

$$\forall r \in \mathrm{dom}(R) \setminus \{0\}\ rR0 \text{ and } \forall s \in \mathrm{dom}(S) \setminus \{0\}\ sS0\ ,$$

then (ω, R) can be embedded orderpreservingly into (ω, S) iff $0 \sim 0$. Since we may simply assume that R and S have maximum elements, this reduces the embeddablility property to the problem of computing $\sim$ with an ITRM. Since S is a wellorder the following lemma yields a recursive definition of $\sim$.

Lemma 2. *For every r and s, $r \sim s$ iff $\forall r' R r \exists s' S s\ r' \sim s'$.*

Proof. Assume $r \sim s$. Take an orderpreserving map

$$\pi : (\mathrm{TC}_R(r), R) \to (\mathrm{TC}_S(s), S) \text{ with } \pi(r) = s.$$

Let $r'Rr$. Let $s' = \pi(r')\ S\ s = \pi(r)$. Then $\mathrm{TC}_R(r') \subseteq \mathrm{TC}_R(r)$ and

$$\pi \restriction \mathrm{TC}_R(r') : \mathrm{TC}_R(r') \to \mathrm{TC}_S(s')$$

orderpreservingly with $\pi(r') = s'$. Thus $\forall r' R r \exists s' S s\ r' \sim s'$.

Conversely assume that $\forall r' R r \exists s' S s\ r' \sim s'$. For every $r'Rr$ choose a map $\pi_{r'} : \mathrm{TC}_R(r') \to \mathrm{TC}_S(s')$ witnessing $r' \sim s'$. Note that

$$\mathrm{TC}_R(r) = \{r\} \cup \bigcup_{r'Rr} \mathrm{TC}_R(r').$$

Thus we may define a map $\pi : \mathrm{TC}_R(r) \to \mathrm{TC}_S(s)$ by $\pi(r) = s$ and for $r'' \neq r$:

$$\pi(r'') = \min\{\pi_{r'}(r'') | r'Rr\}$$

where the minimum is formed with respect to the the wellorder S. Then π witnesses that $r \sim s$.

We shall compute $\sim$ on an ITRM using finite *stacks* of natural numbers. Code a stack $(r_0, \ldots, r_{m-1})$ by $r = 2^{r_0} \cdot 3^{r_1} \cdots p_{m-1}^{r_{m-1}+1}$. Standard stack operations like *pushing* and *popping* natural numbers or finding the *length* $m-1$ of the stack r are recursive and thus computable by an ITRM. Since the relations R and S are infinite time register computable the question whether the stack $(r_0, \ldots, r_{m-1})$ is strictly descending in R or S can also be computed by an ITRM. For the subsequent program we shall use two registers `A` and `B` as stacks with associated operations `pushA`, `popA`, `lenghthA`, `A-is-decreasing-in-R` and `pushB`, `popB`, `lenghthB`, `B-is-decreasing-in-S`. The specific coding of stack contents leeds to a controlled limit behaviour:

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

So let us assume that R and S both have 0 as their maximum element. Running the following program P on an ITRM outputs `yes`/`no` depending on whether R can be embedded order-preservingly into S. 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 `pushA` are understood as *macros* which are inserted into the code with appropriate renaming of variables and statement numbers.

```
      pushA 0;
      pushB 0;
      FLAG := 1; %% ask whether 0 ~ 0
Loop: Case1: if FLAG=0 and lengthA=lengthB=1 %% 0 ~ 0
          then begin; output 'yes'; stop; end;
      Case2: if FLAG=0 and lengthA>lengthB=1 %% 0 !~ 0
          then begin; output 'no'; stop; end;
      Case3: if FLAG=0 and lengthA = lengthB > 1
      %% last element of A ~ last of B
          then begin; %% check next
          popA N;
          pushA N+1;
          popB N;
          pushB 0;
          FLAG:=1; %% ask whether last of A ~ last of B
          goto Loop;
          end;
      Case4: if FLAG=0 and lengthA>lengthB
      %% 2nd-but-last of A !~ last of B
          then begin;
          popA N;
          popB N;
          pushB N+1;
```

```
        FLAG:=1; %% ask whether last element of A ~ last of B
        goto Loop;
        end;
    Case5: if FLAG=1 and A-is-decreasing-in-R
        and B-is-decreasing-in-S
        then begin;
        pushA 0;
        pushB 0;
        FLAG:=0; FLAG:=1; %% flash the flag
        goto Loop;
        end;
    Case6: if FLAG=1 and A-is-decreasing-in-R
        and not B-is-decreasing-in-S
        then begin;
        popB N;
        pushB N+1;
        FLAG:=0; FLAG:=1; %% flash the flag
        goto Loop;
        end;
    Case7: if FLAG=1 and not A-is-decreasing-in-R
        then begin;
        popA N;
        pushA N+1;
        popB N;
        pushB 0;
        FLAG:=0; FLAG:=1; %% flash the flag
        goto Loop;
        end;
```

The next Lemma proves the correctness of the program. Note that the program will always loop back to Loop until the program stops.

Lemma 3. *Let*

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

be the computation by P with trivial input $(0, 0, \ldots)$. Then the computation satisfies:

a) Suppose the machine is in state Loop *and the stack contents of* A *and* B *are $(r_0, \ldots, r_{m-1})$ and $(s_0, \ldots, s_{m-1})$, $m \geqslant 1$ which descend strictly in R and S resp. Moreover suppose that* Flag=1 *and $r_{m-1} \sim s_{m-1}$. 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, $(r_0, \ldots, r_{m-1})$ and $(s_0, \ldots, s_{m-1})$ will always be initial segments of the stacks* A *and* B *resp.*

b) Suppose the machine is in state Loop *and the stack contents of* A *and* B *are $(r_0, \ldots, r_{m-1})$ and $(s_0, \ldots, s_{m-1})$, $m \geqslant 1$ which descend strictly in R and S resp. Moreover suppose that* Flag=1 *and $r_{m-1} \not\sim s_{m-1}$. Let r_m be* *the smallest integer such that $r_m R r_{m-1}$ for which there is no $s_m S s_{m-1}$ such that $r_m \sim s_m$. Then the machine will reach the state* `Loop` *with stack contents $(r_0, \ldots, r_{m-1}, r_m)$ and $(s_0, \ldots, s_{m-1})$ and* `Flag=0` *after a certain interval of time; during that interval, $(r_0, \ldots, r_{m-1})$ and $(s_0, \ldots, s_{m-1})$ will always be initial segments of the stacks* `A` *and* `B` *resp.*

c) *If R can be embedded orderpreservingly into S then the computation stops with output* `'yes'`*.*

d) *If R cannot be embedded orderpreservingly into S then the computation stops with output* `'no'`*.*

Proof. a) and b) are proved by simultaneous induction on s_{m-1} along the well-order S. So consider a situation $(r_0, \ldots, r_{m-1})$ and $(s_0, \ldots, s_{m-1})$ as in a) or b) and assume that a) and b) already hold for all appropriate stacks $(r'_0, \ldots, r'_{m'-1})$ and $(s'_0, \ldots, s'_{m'-1})$ with $s'_{m'-1} S s_{m-1}$.

We first prove a) for the given situation. So `Flag=1` and $r_{m-1} \sim s_{m-1}$. Inspection of the program shows that the machine will successively enter the main loop with register `A` containing the stacks $(r_0, \ldots, r_{m-1}, i)$ for $i = 0, 1, \ldots$. Note that by `Case7`, only the strictly decreasing stacks with $i R r_{m-1}$ are relevant. For such a $(r_0, \ldots, r_{m-1}, i)$ in register `A` the machine will enter the main loop with register `B` containing stacks $(s_0, \ldots, s_{m-1}, j)$. Again, by `Case6`, only strictly decreasing stacks $(s_0, \ldots, s_{m-1}, j)$ with $j S s_{m-1}$ are relevant. In these cases, the main loop is entered with strictly descending stack contents $(r_0, \ldots, r_{m-1}, i)$ and $(s_0, \ldots, s_{m-1}, j)$ and `Flag=1`.

We can apply the inductive assumptions: If $i \sim j$ the machine will subsequently reach the state `Loop` with the same stack contents and `Flag=0`. If $i \not\sim j$ the machine will reach the state `Loop` with stack contents $(r_0, \ldots, r_{m-1}, i, k)$, some $k < \omega$, and $(s_0, \ldots, s_{m-1}, j)$ and `Flag=0`; it will then set the stack contents to $(r_0, \ldots, r_{m-1}, i)$ and $(s_0, \ldots, s_{m-1}, j+1)$ with `Flag=1`. Since $r_{m-1} \sim s_{m-1}$ there is some j such that $i \sim j$ and so the machine will eventually reach the state `Loop` with stack contents $(r_0, \ldots, r_{m-1}, i)$ and $(s_0, \ldots, s_{m-1}, j)$, some $j < \omega$, and `Flag=0`. This will be the case in turn for all $i < \omega$. By the limit rules the limit of these configurations will be a machine configuration with stack contents $(r_0, \ldots, r_{m-1})$ and $(s_0, \ldots, s_{m-1})$, and `Flag=0`.

For b) assume that `Flag=1` and $r_{m-1} \not\sim s_{m-1}$. Let r_m be defined as above. Then the machine will proceed as in the proof of a), until it reaches the stack contents $(r_0, \ldots, r_{m-1}, r_m)$. We argue inductively that it will subsequently set the contents of `B` to $(s_0, \ldots, s_{m-1}, j)$ for $j = 0, 1, \ldots$ and enter the main loop with `Flag=1`.

For $j = 0$, an analysis of the program shows that when the contents of `A` are first set to $(r_0, \ldots, r_{m-1}, r_m)$, the contents of `B` are set to $(s_0, \ldots, s_{m-1}, 0)$ (`Case3` or `Case5`). For the inductive step assume that the machine enters the main loop with stack contents $(r_0, \ldots, r_{m-1}, r_m)$ and $(s_0, \ldots, s_{m-1}, j)$ with `Flag=1`. If $(s_0, \ldots, s_{m-1}, j)$ is not strictly descending in S then `Case6` will modify the contents of `B` to $(s_0, \ldots, s_{m-1})$ and $(s_0, \ldots, s_{m-1}, j+1)$ and enter the main loop with `Flag=1`. If $(s_0, \ldots, s_{m-1}, j)$ is strictly descending in S then we can apply the inductive assumptions. Since $r_m \not\sim j$ the machine will reach the state

`Loop` with stack contents $(r_0, \ldots, r_{m-1}, r_m, k)$, some $k < \omega$, and $(s_0, \ldots, s_{m-1}, j)$ and `Flag=0` after a certain interval of time. Then `Case4` will modify the stack contents to $(r_0, \ldots, r_{m-1}, r_m)$ and $(s_0, \ldots, s_{m-1}, j+1)$, set `Flag:=0` and enter the main loop. This concludes the induction.

By the limit rules the limit of this inductive sequence of configurations will be a configuration with state `Loop`, `Flag=0`, and stack contents $(r_0, \ldots, r_{m-1}, r_m)$ and $(s_0, \ldots, s_{m-1})$, as required by b). Inspection of the algorithm shows that the desired configurations for a) and b) are first reached with the stack contents always endextending $(r_0, \ldots, r_{m-1})$ and $(s_0, \ldots, s_{m-1})$ resp.
c) Assume that R can be embedded orderpreservingly into S. Since 0 is the maximum element of both R and S, $0 \sim 0$. The computation will first reach state `Loop` with stack contents (0) and (0) and `Flag=1`. By a), it will later reach state `Loop` with stack contents (0) and (0) and `Flag=0`. By `Case1` of the main loop, the machine will output `'yes'` and stop.
d) is proved an analogy with c).

4 Admissible Sets and Infinite Register Computations

For the converse we show

Lemma 4. *Let $I : \theta \to \omega, R : \theta \to ({}^{\omega}\omega)$ be a computation by a program P which stops at some successor ordinal $\theta = \beta + 1$. Then $\theta < \omega_1^{\mathrm{CK}}$.*

Proof. Assume that $\theta \geqslant \omega_1^{\mathrm{CK}}$. Let $I(\omega_1^{\mathrm{CK}}) = k$ and

$$R(\omega_1^{\mathrm{CK}}) = (n_0, \ldots, n_{l-1}, 0, 0, \ldots)$$

where $R_0, \ldots, R_{l-1}$ includes all the registers mentioned in the program P. By the liminf rules for ITRMs there is some $\alpha < \omega_1^{\mathrm{CK}}$ such that the sets

$$\{t \in (\alpha, \omega_1^{\mathrm{CK}}) | I(t) = k\}$$

and

$$\{t \in (\alpha, \omega_1^{\mathrm{CK}}) | R_j(t) = n_j\}$$

are closed unbounded in ω_1^{CK}. These sets are Σ_1-definable over the admissible set $L_{\omega_1^{\mathrm{CK}}}$ in the parameter α. In $L_{\omega_1^{\mathrm{CK}}}$ define a sequence $\alpha_0 = \alpha < \alpha_1 < \alpha_2 < \ldots$ such that

$$\exists t \in (\alpha_n, \alpha_{n+1})\ I(t) = k \text{ and for } j = 0, \ldots, l-1 \exists t \in (\alpha_n, \alpha_{n+1})\ R_j(t) = n_j.$$

Such a sequence may be defined by a Σ_1-definition over $L_{\omega_1^{\mathrm{CK}}}$. By the Σ_1-bounding principle in $L_{\omega_1^{\mathrm{CK}}}$, $\alpha^* = \bigcup_{n<\omega} \alpha_n < \omega_1^{\mathrm{CK}}$. Also $I(\alpha^*) = k$ and $R(\alpha^*) = (n_0, \ldots, n_{l-1}, 0, 0, \ldots)$. So the constellation $I(t) = k$ and $R(t) = (n_0, \ldots, n_{l-1}, 0, 0, \ldots)$ occurs at times α^* and ω_1^{CK}. This means that the machine runs into a cycle and does not stop, contrary to our assumption.

Lemma 5. *Let $x \subseteq \omega$ be computable by an infinite time register machine. Then $x \in L_{\omega_1^{\mathrm{CK}}}$.*

Proof. Let P be a register program such that such that for every $n < \omega$

$$P : (n, 0, 0, \ldots) \mapsto \chi_x(n).$$

For $n < \omega$ let the computation by P with input $(n, 0, 0, \ldots)$ stop at time θ_n. By the previous lemma, $\theta_n < \omega_1^{\mathrm{CK}}$. Therefore the computation by P with input $(n, 0, 0, \ldots)$ is an element of $L_{\omega_1^{\mathrm{CK}}}$. The characteristic function χ_x is Δ_1-definable over $L_{\omega_1^{\mathrm{CK}}}$ by

$$\begin{aligned} \chi_x(n) = 1 & \text{ iff there is a computation by } P \text{ with input } (n, 0, 0, \ldots) \text{ and output } 1 \\ & \text{ iff all computations by } P \text{ with input } (n, 0, 0, \ldots) \text{ stop with output } 1. \end{aligned}$$

Since the admissible set $L_{\omega_1^{\mathrm{CK}}}$ satisfies Δ_1-separation, $x \in L_{\omega_1^{\mathrm{CK}}}$.

5 Further Considerations

One may consider variants of the ITRMs, where the registers can hold ordinals below a certain bound β. What is the collection of subsets of β computable by β-ITRMs? It is hoped that such interpolations between ITRMs and ORMs yield a stratification of the constructible sets which may lead to a fine structure theory of the class L of constructible sets (see [5]).

References

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

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

[3] Peter Koepke. Turing computations on ordinals. *Bulletin of Symbolic Logic*, 11(3):377–397, 2005.

[4] Peter Koepke. Computing a model of set theory. In: S. Barry Cooper, Benedikt Löwe, Leen Torenvliet (editors). *New Computational Paradigms: First Conference on Computability in Europe, CiE 2005. Proceedings.* Lecture Notes in Computer Science 3526:223–232, 2005.

[5] Peter Koepke and Sy Friedman. An elementary approach to the fine structure of *L*. *Bulletin of Symbolic Logic*, 3(4):453–468, 1997.

[6] Peter Koepke and Martin Koerwien. Ordinal computations. To appear in: *Mathematics of Computation at CiE 2005.* Special issue of the journal *Mathematical Structures in Computer Science*: 17 pages.

[7] Peter Koepke and Ryan Siders. Computing the recursive truth predicate on ordinal register machines. Submission to CiE 2006, Swansea.

[8] Gerald E. Sacks. *Higher Recursion Theory.* Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1990.