Arch. Math. Logic (2010) 49:249–273
DOI 10.1007/s00153-009-0167-x **Mathematical Logic**

The basic theory of infinite time register machines

Merlin Carl · Tim Fischbach · Peter Koepke · Russell Miller · Miriam Nasfi · Gregor Weckbecker

Received: 18 November 2009 / Accepted: 19 November 2009 / Published online: 12 January 2010

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 register contents are defined by appropriate limit operations. In this paper, we examine the ITRMs introduced by the third and fourth author (Koepke and Miller in Logic and Theory of Algorithms LNCS, pp. 306–315, 2008), where a register content at a limit time is set to the lim inf of previous register contents if that limit is finite; otherwise the register is reset to 0. The theory of these machines has several similarities to the infinite time Turing machines (ITTMs) of Hamkins and Lewis. The machines can decide all Π^1_1 sets, yet are strictly weaker than ITTMs. As in the ITTM situation, we introduce a notion of *ITRM-clockable ordinals* corresponding to the running times of computations. These form a transitive initial segment of the ordinals. Furthermore we prove a *Lost Melody theorem*: there is a real r such that there is a program P that halts

M. Carl · T. Fischbach · P. Koepke · M. Nasfi · G. Weckbecker (✉)
Mathematisches Institut, Universität Bonn, Bonn, Germany
e-mail: gregorw@uni-bonn.de

M. Carl
e-mail: carl@math.uni-bonn.de

T. Fischbach
e-mail: fischbach@uni-bonn.de

P. Koepke
e-mail: koepke@math.uni-bonn.de

M. Nasfi
e-mail: nasfi@cs.uni-bonn.de

R. Miller
Queens College and The Graduate Center, City University of New York, New York, NY, USA
e-mail: Russell.Miller@qc.cuny.edu

on the empty input for all oracle contents and outputs 1 iff the oracle number is r, but no program can decide for every natural number n whether or not $n \in r$ with the empty oracle. In an earlier paper, the third author considered another type of machines where registers were not reset at infinite lim inf's and he called them infinite time register machines. Because the resetting machines correspond much better to ITTMs we hold that in future the *resetting* register machines should be called ITRMs.

Keywords Ordinal computability · Hypercomputation · Infinitary computation · Register machine

Mathematics Subject Classification (2000) 03D60

1 Introduction

The infinite time register machines studied in the present paper were introduced by the third and fourth author in [7]. The aim is to stretch standard register machines into the infinite in a way similar to the ITTMs of Joel D. Hamkins and Andy Lewis [2]: let the "standard hardware" of register machines run in transfinite ordinal time. Successor steps are determined by standard register machine commands. At limit times the register contents are defined using lim inf's of the previous register contents.

A crucial issue is the limit behavior when the lim inf is infinite. In a previous version (see [5]), machines halted or "crashed" on encountering such an overflow; computability by those machines exactly corresponded to hyperarithmetic definitions. We obtain a stronger notion of computability by continuing computations beyond such crashes. So let register machines reset a register to 0 whenever it overflows. This defines richer descriptive classes which are in closer analogy with the ITTM-definable classes. Resetting a register to 0 has some similarities to resetting *Turing* heads to position 0 at limit times. The added strength and the similarities to ITTMs motivate us to propose the name *infinite time register machine* for the machines in this paper. The machines defined in [5] by the third author could be called *non-resetting infinite time register machines*.

Indeed, ITRMs are strictly weaker then ITTMs: the halting problem for ITRMs can be decided by an ITTM (see Theorems 3 and 4 of [7] and also Theorem 4 of the present paper). This illustrates the phenomenon that notions of computability that have the same strength in the finite domain may differ markedly in the infinite. The classical equivalence of register and *Turing* computability rests on the fact that a finite tape inscription can be coded by a single integer, using standard arithmetical operations. In contrast the tape contents of an ITTM cannot in general be coded by finitely many integers or register contents. This explains the power of ITTMs in comparison with ITRMs.

A decisive component for the behavior of infinitary machines are the limit rules for updating the machine configuration at limit times. The ITRMs considered in the present paper are strictly stronger than the register machines in [5] because the present limit rule can be used to test for wellfoundedness (see Theorem 1). Infinitary register and *Turing* machines may, however, converge if "space" and "time" are increased to some admissible ordinal α or to the class Ord of all ordinal numbers (see [6,8,9]).

The second section of the present paper contains basic definitions and a review of the results of [7]. In analogy with the theory of ITTMs we introduce a notion of *ITRM-clockable ordinals* corresponding to the running times of computations. A difference to the theory of ITTMs can be seen in the halting behavior of ITRMs: the set of all ITRM-clockable ordinals is an initial segment of the ordinals, whereas the ITTM-clockable ordinals have gaps. We also present a relation to computable ordinals. In the final section we transfer the “lost melody” theorem for ITTMs to ITRMs: there is a “lost melody” real r and a program P that halts on the empty input for all oracle contents and outputs 1 iff the oracle number is r, while on the other hand no program with the empty oracle can decide for every natural number n whether or not $n \in r$.

Ongoing work by the third author is aimed at determining exact strengths of ITRMs for all numbers N of registers.

2 Infinite time register machines

Definition 1 Let N be a natural number. An N*-register machine* has registers $R_0, R_1, \ldots, R_{N-1}$ which can hold natural numbers. An N*-register 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 where m, n range over the numbers $0, 1, \ldots, N-1$:

a) the *zero instruction* $Z(n)$ changes the contents of R_n to 0;
b) the *successor instruction* $S(n)$ increases the natural number contained in R_n by 1;
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;
e) the *jump instruction* $J(m, n, q)$ is carried out as follows: the contents r_m and r_n of the registers R_m and R_n are compared; then, if $r_m = r_n$, the machine proceeds to the qth instruction of P; if $r_m \neq r_n$, the machine proceeds to the next instruction in P.

The instructions of the program can be addressed by their indices which are called *program states*. At 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 $R(\tau){:}N \to \omega$. $R(\tau)(n)$ is the content of register R_n at time τ. We also write $R_n(\tau)$ instead of $R(\tau)(n)$.

Definition 2 Let P be an N-register program. Let $R_0(0), \ldots, R_{N-1}(0)$ be natural numbers and $Z \subseteq \omega$ be an oracle. These data determine the *infinite time register computation*

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

with program P, input $R_0(0), \ldots, R_{N-1}(0)$ and oracle Z by recursion:

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 $\tau < \theta$ 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 $I(\tau+1)$, $R(\tau+1)$ is determined by the instruction $I_{I(\tau)}$ according to the previous definition;
e) If $\tau < \theta$ is a limit ordinal, then $I(\tau) = \liminf_{\sigma \to \tau} I(\sigma)$ and for all $k < \omega$

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

By the second clause in the definition of $R_k(\tau)$ the register is *reset* in case $\liminf_{\sigma\to\tau} R_k(\sigma) = \omega$. We also write $R_k(\tau)\uparrow$ to distinguish this situation from the case where a register is assigned 0 by the first clause in the definition of $R_k(\tau)$.

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 A partial function $F{:}\omega^n \rightharpoonup \omega$ is *computable* if there is some N-register program P such that for every n-tuple $(a_0, \ldots, a_{n-1}) \in \mathrm{dom}(F)$,

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

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* in the oracle Y if there is some N-register program P such that for all $Z \subseteq \omega$:

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

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

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

In finite time, register computability and *Turing* computability coincide. In transfinite time there are several parallels between ITRMs and ITTMs [7]:

Theorem 1 *The set* WO $= \{Z \subseteq \omega \mid Z$ *codes a wellorder*$\}$ *is computable by an* ITRM *in the empty oracle* $\emptyset$. *The decision procedure described in* [7] *will run at least* α *steps to accept a wellorder* Z *of ordertype* α.

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

Further parallels are proved in the present paper.

On the other hand there are marked differences between registers and tapes. At any point in time register contents are a finite tuple of natural numbers whereas tape contents can be viewed as a function from ω to 2. This makes ITTMs considerably stronger so that they are able to solve halting problems for ITRMs.

Theorem 3 *Let*

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

be the infinite time register computation by P with input $(0, 0, \dots, 0)$ *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*

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

Theorem 4 *For every $N < \omega$ the restricted halting problem*

$$H_N{:} = \{(P, Z) \mid P \textit{ is an } N\textit{-register program, } Z \subseteq \omega\textit{, and the computation by } P \textit{ with input } (0, 0, \dots, 0) \textit{ and oracle } Z \textit{ halts}\}$$

is decidable by some ITRM *with empty oracle.*

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

3 Clockable ordinals

The study of ITRM-computation leads to the question how long a halting computation exactly takes. In contrast to the standard setting, where obviously all natural numbers are possible running times, we anticipate some difficulties. Since there are only countably many ITRM-programs there can only be countably many possible running times. The question of the structure of the class of possible running times will guide us through this and the following chapter.

We call an ordinal α *clockable*, if we can find a program P which performs α many steps and then halts. Measuring the lengths of computations we want to ignore the final halt instruction. This way we obtain a precise notion of clockable limit ordinals even though computation lengths always have to be successor ordinals (see the discussion at the end of Definition 2). This leads to the following technical definition:

Definition 5 An ordinal $\alpha \in \mathrm{Ord}$ is (ITRM-)*clockable* iff there exists an N-register ITRM-program P and a halting computation on input $(0, \dots, 0)$ with empty oracle $\emptyset$ of the form

$$I : \alpha + 2 \to \omega, \quad R : \alpha + 2 \to ({}^{N}\omega)\,.$$

CLOCK $:= \{\alpha \mid \alpha$ clockable$\}$ denotes the set of all clockable ordinals.

Any natural number $n \in \omega$ is clockable because $n + 1$ consecutive zero instructions constitute a program that states the clockability of n.

A program witnessing that ω is clockable uses a flag which is initialized by 1 and which is switched from 1 to 0 and back to 1 in each iteration. By the lim inf-rule of

Definition 2e) the value of the flag becomes 0 after ω many steps. A program that halts when the value of the flag becomes 0 clocks ω. By using n flags one is easily convinced that ω^n with $n \in \omega$ is clockable. This observation implies:

Proposition 1 *All $\alpha < \omega^\omega$ are clockable.*

ω^ω is equal to $\sup\{\omega^n \mid n \in \omega\}$. We want to use this characterization to prove the following:

Lemma 1 *ω^ω is clockable.*

To prove this, we store a finite number of bits in one register such that they behave in a reasonable way at limit times. For $n \in \omega$ define the function $\rho_n \colon {}^n 2 \to \omega$:

$$\langle b_i | i \in n\rangle \longmapsto \sum_{i \in n} b_i \cdot 2^i$$

All ρ_n are register computable functions and there are register computable projection functions π_n with $\pi_n(\rho_n(\langle b_i | i \in n\rangle), \ell) = b_\ell$. Also the functions σ_n with $\sigma_n(\langle b_0, \dots, b_{n-1}\rangle, \ell, b) = \rho_n(\langle b_0, \dots, b_{\ell-1}, b, b_{\ell+1}, \dots, b_{n-1}\rangle)$ are register computable.

Proposition 2 *Let $\lambda \in \mathrm{Lim}$ and $n \in \omega$. Let $\left\langle \left\langle b_i^\zeta | i \in n \right\rangle | \zeta \in \lambda \right\rangle$ be a sequence with $b_i^\zeta \in 2$. Let $i \in n$ be maximal, such that there is a $\alpha < \lambda$, such that for all $i < j < n$ and for all $\alpha \le \zeta < \lambda$, b_j^ζ is constant. Then for all $i \le \ell < n$:*

$$\pi_n\left(\liminf_{\zeta \to \lambda} \rho_n\left(\left\langle b_i^\zeta | i \in n \right\rangle\right), \ell\right) = \liminf_{\zeta \to \lambda} b_\ell^\zeta$$

Proof Choose i as above. W.l.o.g. we may assume that $i = n - 1$.

Case 1: $\liminf_{\zeta \to \lambda} b_{n-1}^\zeta = 1$. But then there is a ordinal α, such that for every $\lambda > \zeta > \alpha$ we have: $b_{n-1}^\zeta = 1$. So $\rho_n\left(\left\langle b_i^\zeta | i \in n \right\rangle\right) \ge 2^{n-1}$. This implies that

$$\liminf_{\zeta \to \lambda} \rho_n\left(\left\langle b_i^\zeta | i \in n \right\rangle, n-1\right) \ge 2^{n-1}$$

holds. But then

$$\pi_n\left(\liminf_{\zeta \to \lambda} \rho_n\left(\left\langle b_i^\zeta | i \in n \right\rangle, n-1\right), n-1\right) = 1$$

holds as well.

Case 2: $\liminf_{\zeta \to \lambda} b_{n-1}^\zeta = 0$. This case is handled in analogy to case 1. □

Proof (Lemma 1) In the following program, we use v as a vector of length n. All vector operations are understood as macros, which have to be substituted by corresponding register programs. The following pseudo code has to be transformed into a valid program P.

```
01 v = 1
02 n = 1
03 while not n = 0 do:
04   for i = n-1, ... , 0 do:
05     if v[i] = 0 do:
06     if i = n-1 do:
07       n = n + 1
08     end if
09     v[i+1] = 0
10     v[i+1] = 1
11     v[i] = 1
12    end if
13  end for
14  v[0] = 0
15  v[0] = 1
16 end while
```

Let

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

denote the computation of the N-register program above on input $(0, \dots, 0)$ and empty oracle. First we note:

(1) *For a limit ordinal $\lambda \in \theta$ we have: $I(\lambda) = 03$.*

Proof This is the case because 03 is the instruction with the smallest number in the outer loop. The inner loop is finite because every manipulation of v is computable by an register machine. The program has to traverse the outer loop after finitely many steps. So 03 is the smallest instruction that is executed unboundedly often. *qed (1)*

For $\alpha \in \theta$ let $v(\alpha)$ denote the value of the register in which v is stored and $n(\alpha)$ denote the value of the register in which n is stored. For a successor ordinal $\alpha \in \theta$ and $1 \leq i < n(\alpha)$ let $v(\alpha)(i) = \pi_{n(\alpha)}(v(\alpha), i)$ and for limit ordinals $\lambda \in \theta$ let $v(\lambda)(i) = \liminf_{\alpha \to \lambda} \pi_{n(\alpha)}(v(\alpha), i)$.

(2) *If $v(\lambda)(i) = 0$ at a limit ordinal $\lambda < \omega^{\omega}$ with $i \geq 1$, then $v(\lambda + \omega^{i})(i) = 0$ and for all limit ordinals β with $\lambda < \beta < \lambda + \omega^{i}$ and $j \geq i$ we have $v(\beta)(j) = 1$.*

Proof We show this result by induction over $i \geq 1$.
Let $i = 0$. It is clear that $v(\alpha)(0) = 0$ for every limit ordinal α, because in every traversal of the main loop we flash the flag. In this case the second part of the proposition is clear.
Let $j = i + 1$. Let α be a limit ordinal with $v(\alpha)(i + 1) = 0$. By the induction hypothesis, for every $\ell \in \omega$, $v(\alpha + \omega^{i} \cdot \ell)(i) = 0$. Because of the lines 05 to 13 the flag stored in $i + 1$-th cell is flashed after every limit ordinal of the form $\alpha + \omega^{i} \cdot \ell$ for $\ell \in \omega$. By the second part of the induction hypothesis, for all limit ordinals $\alpha < \beta < \alpha + \omega^{i+1}$ and $j \geq i$, we must have $v(\beta)(j) = 1$, so Proposition 2 shows that we take the lim

inf. Hence $v(\alpha + \omega^i \cdot \ell)(i+1) = 0$. The second part of the proposition follows from the fact that a flag can be 1 at limit times iff it is flagged unbounded often. But for $j \geq i$ this is not the case. *qed* (*2*)

Now we make the following observation:

(3) *For every $m \in \omega$ there is $\ell \in \omega$ such that $m = n(\omega^m + \ell)$ and for all $\alpha < \omega^m n(\alpha) \leq m$.*

and

(4) *If for a limit ordinal $\lambda \in \theta$ and some $1 \leq i < n(\lambda)$ we have $v(\lambda)(i) = 0$, then also $v(\lambda)(j) = 0$ whenever $1 \leq j < i$.*

At one limit stage n is reset to 0 and by (1) the current instruction at this time is 03. But then the program stops.
According to (3) n is bounded below ω^m. Because n is never decreased it cannot reach zero before ω^ω. n grows unbounded below ω^ω and so the register is reset. By (1) the instruction 03 is executed and so the program stops. Hence it clocks ω^ω. □

4 The structure of CLOCK

To get a deeper insight into the structure of CLOCK we first prove a speed-up lemma, that we will generalize later. The term "speed-up" is adopted from the analogous *infinite time Turing machine* results. Here it seems more appropriate to speak of a truncation lemma.

Lemma 2 (Speed-up) *Let $\alpha + n$ be a clockable ordinal for some $n \in \omega$. Then α itself is* ITRM-clockable.

Proof Following the strategy of the corresponding proof for ITTMs (see [2]), we design an algorithm which uses additional registers to anticipate the stage α configuration. We thereby obtain a flag indicating that the registers are set up in a way which will cause the machine to stop in n additional steps.

Since every computation can easily be made longer by a finite amount of steps, we can assume that α is a limit ordinal. Since $\alpha + n$ is clockable, there is an ITRM N-register program P with a computation

$$I{:}\alpha + n + 2 \to \omega, \quad R{:}\alpha + n + 2 \to ({}^N\omega)$$

by P on input $(0, \dots, 0)$ with empty oracle. At stage α the machine is in a configuration $(\bar{I}, \bar{R}) := (I(\alpha), R(\alpha))$ which will cause it to halt after n additional steps. Therefore stage α has to be the first (and only) occurrence of configuration $(\bar{I}, \bar{R})$ in the computation. We modify P in a way which lets us decide in one step whether configuration $(\bar{I}, \bar{R})$ has been entered. This will result in a computation of length $\alpha + 2$.

The computation by P involves only finitely many registers. Let $S \subset \omega$ be the set of indices of those registers. Let $X \subset S$ contain the indices i of registers with $R_i(\alpha) \uparrow$.

Finally let $N := S \setminus X$. The following program is designed to handle the case $X \neq \emptyset$. The case $X = \emptyset$ can be solved by a much simpler program as noted below.

We fix four registers, denoted by R_{A}, $R_{\min}$, R_{cmin}, R_{flag}, which are not used by P. R_{A} is used as an array of bits to keep track of registers in N, which have attained the correct value from $\bar{R}$. We want $R_{\min}$ to always contain the minimum of all registers in X. R_{cmin} shall contain a cumulative minimum, i.e., the smallest value of $R_{\min}$ since R_{cmin} was reset for the last time. R_{flag} shall be 0 iff configuration $(\bar{I}, \bar{R})$ is entered.

We modify P in three steps and obtain a program P'.

1. Prepend an instruction to P which initializes R_{flag} to 1.
2. Before the $\bar{I}$-th instruction of P insert the following block of instructions.

```
CHECK-FLAG:
   01 if FLAG = 0
   02   stop
   03 end
```

3. Before each instruction of P insert the following block of instructions named `UPDATE`.

Here the registers R_{A}, $R_{\min}$, R_{cmin}, R_{flag} are denoted by `A`, `MIN`, `CMIN`, `FLAG`. For $i < \omega$ the i-th register is referred to as `R[i]`. For $n \in N$ the value of $\bar{R}_n$ is written as `r_n`. Let $X = \{$n_0, . . . ,n_k$\}$.

```
UPDATE:
 A01  MIN = R[n_0]
 [For i = 1, ... , k]
  A02  if MIN > R[n_i]
  A03    MIN = R[n_i]
  A04  end
 ...
  B01  if CMIN > MIN
  B02    CMIN = MIN
  B03  end
 [For each n in N]
  C01  if R[n] < r_n
  C02    FLAG = 1
  C03  elseif R[n] = r_n
  C04    A[n] = 1
  C05  end
 ...
  D01  if all-components-one(A)
  D02    A = 0
  D03    FLAG = CMIN + 2
  D04    CMIN = MIN
  D05  end
```

Clearly the jump instructions in P have to be updated. The first lines of the inserted blocks `UPDATE` and `CHECK-FLAG` respectively shall be used as new jump destinations.

If $X = \emptyset$ most of this bookkeeping can be avoided: Simply initialize a flag to 2. Set it to 1 and back to 2 whenever the situation `D02-D04` is encountered. Set it to 0 and back to 2 whenever a register falls below its designated value from $\bar{R}$. An analogous argument as below shows that at limit times the flag is 1 if and only if the register values from $\bar{R}$ are attained. We concentrate on the more complicated case $X \neq \emptyset$ for the rest of the proof.

Let (I', R') be the computation by P' on input on input $(0, \ldots, 0)$ and empty oracle. Note that the inserted instructions neither contain jump instructions nor modify the register used by P. Therefore the computation by P' can be thought of as an extension of P where single instructions have been replaced by blocks of instructions occurring in the same order. Especially since all additions are finite the configurations of (I', R') at limit stages will be equal to those of (I, R) for registers from S.

(1) *If λ is a limit ordinal then $R_{\min}(\lambda) \uparrow$ iff $\forall n \in X\ R_n(\lambda) \uparrow$.*

Proof `UPDATE` is executed before every instruction of P. Therefore after each possible change (by an instruction or by the liminf rule) of a register in X at stage σ lines `A01-A04` ensure that $R_{\min}$ is set to $\min\{R_n(\sigma) \mid n \in X\}$ after a finite amount of steps before the computation continues. In particular the registers from X remain unchanged until $R_{\min}$ has been updated. Hence (*): Every minimum of registers in X at any stage is assumed by $R_{\min}$ within finitely many subsequent steps.
Let λ be a limit ordinal. Assume for contradiction that $R_{\min}(\lambda) \uparrow$ but $R_n(\lambda) \not\uparrow$ for some $n \in X$. Let $x := R_n(\lambda)$. Then there are cofinally many τ below λ such that $R_n(\tau) = x$ and thus $\min\{R_n(\tau) \mid n \in X\} \leq x$. Because of (*) there are cofinally many τ' below λ such that $R_{\min}(\tau') \leq x$ which implies $R_{\min}(\lambda) \leq x$. Contradiction.
Let $R_n(\lambda) \uparrow$ for all $n \in X$. Then $\liminf_{\tau \to \lambda} \min\{R_n(\tau) \mid n \in X\} = \min\{\liminf_{\tau \to \lambda} R_n(\tau) \mid n \in X\} = \omega$. And since $R_{\min}$ is only updated with values from registers R_n with $n \in X$, $R_{\min}(\lambda) \uparrow$ follows. *qed (1)*

(2) *Let λ be a limit ordinal. $R_{\text{flag}}(\lambda) = 0$ iff $R_n(\tau) = \bar{R}_n$ for all $n \in N$, $R_n(\tau) \not\uparrow$ for all $n \in N$ and $R_n(\tau) \uparrow$ for all $n \in X$.*

Proof Observe $R_{\text{flag}} = 0$ iff $R_{\text{flag}} \uparrow$. Assume $R_n(\lambda) < \bar{R}_n$ for some $n \in N$. Let $x := R_n(\lambda)$. Then $R_n(\tau) = x$ for cofinally many τ below λ. Then R_{flag} will be set to 1 cofinally often below λ (line `C02`). Hence $R_{\text{flag}}(\lambda) \not\uparrow$.
Assume $R_n(\lambda) > \bar{R}_n$ or $R_n(\lambda) \uparrow$ for some $n \in N$. Then R_n will be assigned $\bar{R}_n$ only boundedly many times in λ. Hence the component of R_{A} corresponding to n will equal 0 at all sufficiently large stages below λ. So R_{flag} will no longer be updated with values from R_{cmin} (line `D03`). In particular $R_{\text{flag}}(\lambda) \not\uparrow$.
Assume $R_n(\lambda) \not\uparrow$ for some $n \in X$. Then $R_{\min}(\lambda) \not\uparrow$ by (1). Hence there are cofinally many τ below λ such that $R_{\min}(\tau) = x := R_{\min}(\lambda)$. The subsequent `UPDATE` block will ensure $R_{\text{cmin}} \leq R_{\min}$. Observe that R_{cmin} is decreasing as long as it is not reset in line `D04`. We can assume that R_{flag} will be updated with values from R_{cmin} cofinally often below λ (else $R_{\text{flag}} \not\uparrow$). So when R_{flag} is updated for the next time, R_{cmin} will not have been reset. So $R_{\text{flag}}(\tau) \leq x + 2$ for cofinally many τ below λ. Hence $R_{\text{flag}}(\lambda) \not\uparrow$.
For the other direction let $R_{\text{flag}}(\lambda) \uparrow$. In particular R_{flag} has been updated cofinally often which implies $R_n(\lambda) \leq \bar{R}_n$ and $R_n(\lambda) \not\uparrow$ for all $n \in N$. Since R_{flag} has not been reset to 1 cofinally often (line `C02`) we also know $R_n(\lambda) \geq \bar{R}_n$.

Note that all values > 1 of R_{flag} correspond to values of R_{cmin}. Since R_{cmin} decreases between these assignments to R_{flag}, $R_{\text{flag}}(\lambda) \uparrow$ implies $R_{\text{cmin}}(\lambda) \uparrow$. For the same reason it follows that $R_{\min}(\lambda) \uparrow$, which in turn by (1) implies $R_n(\lambda) \uparrow$ for all $n \in X$. *qed* (*2*)

(3) $\text{dom}(I') = \alpha + 2$.

Proof Assume $\alpha \in \text{dom}(I')$. Since the behaviour of P' on registers from S corresponds to P and α is limit ordinal: $R'_n(\alpha) = \bar{R}_n$ for all $n \in N$ and $R'_n(\alpha) \uparrow \leftrightarrow \bar{R}_n \uparrow$ for all $n \in S$. For the same reason $I'(\alpha)$ is the first instruction of `CHECK-FLAG`. Then by (2): $R_{\text{flag}}(\alpha) = 0$ and the next instruction will be the halt instruction. On the other hand the computation by P' can not stop before α, because α is the first occurence of the configuration $(\bar{I}, \bar{R})$. So by (2) R_{flag} will not be zero and the (enhanced) computation will proceed like P. *qed* (*3*)

Hence α is clockable. □

We want to generalize the Speed-up Lemma. For this we state another halting criterion.

Theorem 5 (Halting Criterion) *Let*

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

be an ITRM*-computation by* P *with input* $(0, \dots, 0)$ *and empty oracle* $\emptyset$*. This computation does not stop iff there is some configuration* $(\bar{I}, \bar{R})$ *such that*

$$\text{otp}(\{t < \theta \mid (I(t), R(t)) = (\bar{I}, \bar{R})\}) \geq \omega^\omega .$$

Proof If the computation does not stop then at least one of the configurations has to occur class-many times. For the converse assume that $(t_i | i \leq \omega^\omega)$ is a strictly increasing and continuous sequence of ordinals $< \theta$ such that

$$\forall i < \omega^\omega (i \notin \text{Lim} \to (I(t_i), R(t_i)) = (\bar{I}, \bar{R})) .$$

By the definition of $(I(t_{\omega^\omega}), R(t_{\omega^\omega}))$ using lim inf operations, there must be some ordinal $\omega^n < \omega^\omega$ such that:

(1) *there is a time* $t \in [t_1, t_{\omega^n})$ *such that* $I(t) = I(t_{\omega^\omega})$*, and for every register index* $k < \omega$ *there is a time* $t \in [t_1, t_{\omega^n})$ *such that* $R_k(t) = R_k(t_{\omega^\omega})$*;*

(2) *for every time* $t \in [t_{\omega^n}, t_{\omega^\omega})$ *the configuration* $(I(t), R(t))$ *will be pointwise* $\geq$ *than the configuration* $(I(t_{\omega^\omega}), R(t_{\omega^\omega}))$.

We claim

(3) $(I(t_{\omega^{n+1}}), R(t_{\omega^{n+1}})) = (I(t_{\omega^\omega}), R(t_{\omega^\omega}))$.

Proof $\geq$ (pointwise) follows from (2).

For the converse observe that for every $m < \omega$

$$(I(t_{\omega^n \cdot m+1}), R(t_{\omega^n \cdot m+1})) = (\bar{I}, \bar{R}) .$$

Then the computations on the time intervals $[t_{\omega^n \cdot m+1}, t_{\omega^n \cdot m+1} + (t_{\omega^n} - t_1))$ all begin with the same configuration and are therefore isomorphic up to a shift in time. By (1), every state and every register content of the configuration $(I(t_{\omega^\omega}), R(t_{\omega^\omega}))$ will be reached during the interval $[t_{\omega^n \cdot m+1}, t_{\omega^n \cdot m+1} + (t_{\omega^n} - t_1))$. Since these intervals limit up to $t_{\omega^{n+1}}$ and by the lim inf rule, the equality is established. *qed* (*3*)

(3) and (2) imply that the computation will cycle from $t_{\omega^{n+1}}$ onwards with a period of length $t_{\omega^\omega} - t_{\omega^{n+1}}$. Thus the computation does not stop. □

With the help of the halting criterion we are now able to prove the generalized Speed-up lemma:

Lemma 3 (Generalized speed-up) *Let α be a clockable ordinal and $\beta < \alpha$. Then β is* ITRM-*clockable.*

As an implication of this lemma we get:

Theorem 6 **(Initial segment)** CLOCK $:= \{\alpha \mid \alpha$ ITRM-*clockable*$\}$ *is a transitive initial segment of the countable ordinals.*

This theorem states a fundamental difference between the theory of ITTM-clockable ordinals (as it is developed in [2]) and ITRM-clockable ordinals. In contrast to the ITTM situation there are no gaps. This can be interpreted as some kind of weakness of ITRMs. We now prove the lemma:

Proof (Generalized Speed-up) Let $I{:}\alpha+2 \to \omega,\ R{:}\alpha+2 \to ({}^N\omega)$ be a computation by a program $P = P_0, \ldots, P_{k-1}$ on input $(0, \ldots, 0)$ and empty oracle. W.l.o.g. β is a limit ordinal. At stage β the machine is in some configuration $(\bar{I}, \bar{R}) :\, = (I(\beta), R(\beta))$. Let

$$M{:} = \{\gamma \mid \gamma \leq \beta \wedge (I(\gamma), R(\gamma)) = (\bar{I}, \bar{R})\}, \quad \eta := \mathrm{otp}(M).$$

We shall simulate the program P while counting the number of times that the configuration $(\bar{I}, \bar{R})$ appears, halting at the ηth time.

By Theorem 5 $\eta < \omega^\omega$. η is clockable by Proposition 1. Since β is a maximal element of the set on the right side η has to be a successor ordinal: $\eta = \eta' + 1$. By Lemma 2 η' is clockable: Let $Q = Q_0, \ldots, Q_{l-1}$ be a N_Q-register program with a computation $I_Q : \eta' + 2 \to \omega,\ R_Q : \eta' + 2 \to ({}^{(N_Q)}\omega)$ by Q on input $(0, \ldots, 0)$ and empty oracle.

We can assume that P and Q use disjoint sets of registers. W.l.o.g. the first m registers are used by P and the registers with indices in $[m, m+n)$ are used by Q. We fix another register R_{line} which is used by neither P nor Q and will be employed as a pointer to the lines in the program Q.

Consider the $N_{P'}$-register program $P' := \tilde{P}_0, \ldots, \tilde{P}_{\bar{I}-1}, B, \tilde{P}_{\bar{I}}, \ldots, \tilde{P}_{k-1}$ where

1. B is the following block of instruction. Again the i-th register is denoted as `R[i]`, $\bar{R}_i$ is referred to by `r_i`, and R_{line} is written as `LINE`:

```
if R[0]= r_0 and R[1] = r_1 and ... and
  R[m] = r_m then
   if LINE = 0
     Q_TILDE_0
   else if LINE = 1
     Q_TILDE_1
   ...
   else if LINE = l - 1
     Q_TILDE_(l-1)
   end

   if LINE >= l
     stop
   end
end
```

`Q_TILDE_i` for $i \in l$ is a placeholder for the following block of instructions:

$$\tilde{Q}_i := \begin{cases} Q_i,\ \texttt{LINE = i + 1}, & \text{if } Q_i \neq J(u,v,q) \\ \texttt{if R[u] = R[v] then} & \text{if } Q_i = J(u,v,q) \\ \quad\texttt{LINE = q} & \\ \texttt{else} & \\ \quad\texttt{LINE = i + 1} & \end{cases}$$

Let $\tilde{q}_i$, $i \in l$, and $\tilde{b}$ be the index of the first instruction of $\tilde{Q}_i$ and B in P' resp.

2. $\tilde{P}_i$ for $i \in k$ is the same instruction as P_i with indices in jump instructions shifted appropriately: For $i \in k$

$$\tilde{P}_i = \begin{cases} P_i, & \text{if } P_i \neq J(u,v,q), \\ J(u,v,q), & \text{if } P_i = J(u,v,q) \text{ and } q \leq \bar{I}, \\ J(u,v,q+|B|) & \text{if } P_i = J(u,v,q) \text{ and } q > \bar{I}. \end{cases}$$

Note that jump instructions with destination $P_{\bar{I}}$ are altered to point to the first line of B.

Let $I' : \alpha' \to \omega,\ R' : \alpha' \to ({}^{(N_{P'})}\omega)$ be the computation by P' on input $(0, \ldots, 0)$ and empty oracle and $\langle t_\tau \mid \tau < \theta \rangle$ be the strictly increasing sequence of stages $\gamma < \alpha'$ such that $I'(\gamma) = \tilde{q}_i$ for some $i \in l$ (i.e. the stages of (I', R') just before an instruction of Q is executed). Define $R'_Q : \alpha' \to ({}^{(N_{P'})}\omega)$ to be the restriction of R' to registers used by Q: $R'_Q(\tau)(i) := R'(\tau)(i)$ when $\tau < \alpha'$ and $i \in [m, m+n)$, and $R'_Q(\tau)(i) = 0$ otherwise.

(1) *If $\tau < \theta$ and $R'_{\text{line}}(t_\tau) = k < l$, then $I'(t_\tau) = \tilde{q}_k$. Moreover, if Q_k is not a jump instruction then $P'_{I'(t_\tau)} = Q_k$.*

Proof By the construction of P'. *qed (1)*

(2) *If* $\tau < \theta$ *then* $R'_Q(t_\tau) = R_Q(\tau)$ *and* $R'_{\text{line}}(t_\tau) = I_Q(\tau)$. *In particular* $\theta \leq \eta$.

Proof By induction on $\tau < \theta$.
Let $\tau = 0$: $R'_Q(t_0) = (0, \dots, 0) = R_Q(0)$ and $R'_{\text{line}}(t_0) = 0 = I_Q(0)$ since neither a register used by Q nor R'_{line} is altered before t_0.
Let $\tau = \nu + 1 < \theta$ and assume the statement is already proved for ν. By the induction hypothesis $R'_Q(t_\nu) = R_Q(\nu)$ and $R'_{\text{line}}(t_\nu) = I_Q(\nu)$.
We can assume $R'_{\text{line}}(t_\nu) = I_Q(\nu) < k$. Otherwise the computation would have halted after t_ν making t_ν the last entry of the sequence. In particular this shows $\theta \leq \text{dom}(I_Q) - 1 = \eta$.
If $I_Q(\nu)$ is a jump instruction then $R_Q(\nu) = R_Q(\nu + 1)$. In this case also $\tilde{Q}_{I_Q(\nu)}$ does not contain write instructions. Hence $R'_Q(t_{\nu+1}) = R'_Q(t_\nu)$ and so $R'_Q(t_\tau) = R_Q(\tau)$ by induction. The second part of $\tilde{Q}_{I_Q(\nu)}$ then sets $R_{\text{line}}(t_{\nu+k})$ for some small k to $I_Q(\nu + 1)$ by simulating the behavior of a jump instruction. This value is not changed until $t_{\nu+1}$ and so $R_{\text{line}}(t_\tau) = I_Q(\tau)$.
If $I_Q(\nu)$ is not a jump instruction then by (4) and the induction hypothesis $P'_{I'(t_\nu)} = Q_{I_Q(\nu)}$. So $R'_Q(t_\nu + 1) = R_Q(\nu + 1)$. After that R'_Q remains constant until t_τ. So $R'_Q(t_\tau) = R_Q(\tau)$. Since $I_Q(\nu)$ is not a jump instruction, $I_Q(\nu + 1) = I_Q(\nu) + 1$. Analogously the second part of $\tilde{Q}_{I_Q(\nu)}$ ensures $R'_{\text{line}}(t_\nu + 2) = I_q(\nu) + 1$. Finally $R'_{\text{line}}(t_{\nu+1}) = R'_{\text{line}}(t_\nu + 2)$ since R_{line} remains constant until $t_{\nu+1}$.
Let $\tau < \theta$ be a limit ordinal. By the induction hypothesis $R'_Q(t_\nu) = R_Q(\nu)$ and $R'_{\text{line}}(t_\nu) = I_Q(\nu)$ for all $\nu < \tau$. Let $\lambda = \sup\{t_\nu \mid \nu < \tau\}$. Since the sequence is strictly increasing, λ is a limit ordinal. Let $i \in [m, m + n)$. Then

$$\begin{aligned} R'_i(\lambda) &= \liminf_{\sigma \to \lambda} R'_i(\sigma) \\ &= \liminf_{\nu \to \tau} R'_i(t_\nu) \\ &= \liminf_{\nu \to \tau} R_{Q,i}(\nu) = R_{Q,i}(\tau). \end{aligned}$$

Here the second equivalence holds because the values taken by $R'_i(\sigma)$ for $\sigma < \lambda$ appear in $R'_i(t_\nu)$ for $\nu < \tau$ in same order and the sequence $\langle t_\nu \mid \nu < \tau \rangle$ is cofinal in λ. Obviously $\lambda \leq t_\tau$. Moreover, R'_i remains unchanged on $[\lambda, t_\tau]$, so we get $R'_i(t_\tau) = R'_i(\lambda) = R_{Q,i}(\tau)$.
For the same reason:

$$\begin{aligned} R'_{\text{line}}(t_\tau) = R'_{\text{line}}(\lambda) &= \liminf_{\sigma \to \lambda} R'_{\text{line}}(\sigma) \\ &= \liminf_{\nu \to \tau} R'_{\text{line}}(t_\nu) \\ &= \liminf_{\nu \to \tau} I_Q(\nu) = I_Q(\tau) \end{aligned}$$

qed (2)

Let $\langle s_\tau \mid \tau < \sigma \rangle$ be the increasing sequence of stages $\gamma < \alpha'$ with $P'_{I'(\gamma)} = \tilde{P}_i$ for some $i \neq \bar{I}$ or $I'(\gamma) = \tilde{b}$.
Since B is traversed in a linear fashion, a similar argument as above shows, that P' behaves essentially like P on the registers used by P. Specifically, the register values at limit stages coincide. Let R'_P again denote the restriction of R' to P's registers.

(3) *If* $\tau < \sigma$ *then* $R'_P(s_\tau) = R(\tau)$. *For limit ordinals* $\lambda < \sigma$*:* $s_\lambda = \lambda$.

In particular (I', R') only halts before β if the stop instruction from B is encountered. Since the block B is inserted before the instruction corresponding to $\bar{I}$, the outer if-condition in B is true exactly at stages s_γ with $(I(\gamma), R(\gamma)) = (\bar{I}, \bar{R})$. At stage β the configuration $(\bar{I}, \bar{R})$ occurs for the ηth time. This implies $\theta = \eta$ and $\alpha' = t_{\eta'} + m$ for some $m < |B|$.

For each t_τ, $\tau < \eta$, there has to exist a unique $\gamma \in M$ such that $s_\gamma < t_\tau < s_\gamma + |B|$. Since β is the maximal element of M and $\eta = \text{otp}(M)$, we get $s_{\eta'} = s_\beta$. Hence $\alpha' = t_\beta + m' = \beta + m'$ for some $m' < 2|B|$. By the Speed-up Lemma 2, β is clockable. □

Lemma 4 *Every recursive ordinal is* ITRM-clockable.

Proof Let α be a recursive ordinal. α then is Turing computable by a program P. Let Z be the result of the computation of P used as an oracle. Instead of consulting an oracle it is also possible to rerun P. We use the program which was introduced in [7] to compute WO (see also Theorem 1). Using recalculated values of Z this program will take at least α many steps to halt. Because the set of ITRM-clockable ordinals is an initial segment of the countable ordinals, α is clockable. □

5 Computable ordinals

In Definition 4, we introduced a notion of *computable real numbers*. With this we can define *computable ordinals*:

Definition 6 An ordinal number α is (*ITRM-*) *computable* if there is a *computable* real number $x \subseteq \omega$ coding a wellorder with

$$\alpha = \text{otp}(\{(a, b) \in \omega \times \omega \mid p(a, b) \in x\})$$

where p denotes an appropriate pairing function.

Theorem 7 *The class of clockable ordinals* CLOCK *coincides with the class of computable ordinals.*

Proof ($\supseteq$) Let α be a computable ordinal. Let $x \subseteq \omega$ denote a code for a wellorder of ordertype α and P a program that computes x.

We again make use of the program which computes WO as described in [7]. If all oracle instructions are replaced by a rerun of P, then this program will run for at least α many steps. By the Initial Segment Theorem α is clockable.

($\subseteq$) Let P be a program with a computation

$$I : \alpha + 2 \to \omega, R : \alpha + 2 \to ({}^N\omega)$$

by P on input $(0, \dots, 0)$ and empty oracle $\emptyset$.

Define for $\bar{I} \in \omega$ and $\bar{R} \in ({}^N\omega)$:

$$\eta(\bar{I}, \bar{R}) = \mathrm{otp}(\{\beta \in \alpha + 2 \mid (I(\beta), R(\beta)) = (\bar{I}, \bar{R})\})$$

(1) $(\forall \beta \in \alpha + 2)\ \eta(I(\beta), R(\beta)) < \omega^\omega$

Proof P halts and Theorem 5 proves the claim. *qed (1)*

The idea is to order the triples of the form $(\bar{I}, \bar{R}, \gamma)$ with $\gamma < \eta(I, R)$ as follows: $(\bar{I}, \bar{R}, \gamma) < (\bar{I}', \bar{R}', \gamma')$ iff the γ-th occurrence of $(\bar{I}, \bar{R})$ is before the γ'-th occurrence of $(\bar{I}', \bar{R}')$.

For this we define

$$S = \{(\bar{I}, \bar{R}, \gamma) \mid (\exists \beta \in \alpha + 2)(\bar{I}, \bar{R}) = (I(\beta), R(\beta)) \wedge \gamma < \eta(I, R)\}$$

Further we define for $(\bar{I}, \bar{R}, \gamma) \in S$:

$$\delta(\bar{I}, \bar{R}, \gamma) = \min(\{\beta \in \alpha + 2 \mid \gamma = \mathrm{otp}(\{\sigma < \beta \mid (\bar{I}, \bar{R}) = (I(\gamma), R(\gamma))\})\})$$

With this the following definition is natural:

$$(\bar{I}, \bar{R}, \gamma) \prec (\bar{I}', \bar{R}', \gamma') \text{ iff } \delta(\bar{I}, \bar{R}, \gamma) < \delta(\bar{I}', \bar{R}', \gamma')$$

The following claim holds:

(2) $(S, \prec)$ is a wellorder and $\alpha + 2 = \mathrm{otp}((S, \prec))$

Proof To prove this claim we show that δ is an order preserving bijection between S and $\alpha + 2$. It is clear that δ is a function from S to $\alpha + 2$. Let $\beta < \alpha + 2$. The preimage of β is $(I(\beta), R(\beta), \mathrm{otp}(\{\gamma < \beta \mid (I(\beta), R(\beta)) = (I(\gamma), R(\gamma))\}))$. The injectivity of δ is proved analogously. δ is order preserving by definition. *qed (2)*

A triple $(\bar{I}, \bar{R}, \gamma)$ can be coded as a natural number. $\bar{I}$ is a natural number, and to code $\bar{R}$ we observe, that only finitely many registers are used in P. Every $\eta < \omega^\omega$ can be represented as $\eta = \omega^{n-1} \cdot c_{n-1} + \ldots + \omega \ldots c_1 + c_0$ for some $c_{n-1}, \ldots, c_0 \in \omega$.
Define x as the set of codes of elements in $\prec$. We want to prove that x is computable. To construct a program that computes x we have to build a counter which will be able to count an ordinal $\eta < \omega^\omega$ that is coded by a sequence of natural numbers $\langle c_i \mid i \in n \rangle$ as above. For this modify the program in the proof of Lemma 1. We observed that on the first limit time with $v(t)(i + 1) = 0$, $t = \omega^i$ holds. By the main claim in this proof we further know that for $\ell \in \omega$ and $t = \omega^i \cdot \ell$, $v(t)(i) = 0$ holds. For every limit time

t with $\omega^i \cdot \ell < t < \omega^i \cdot (\ell + 1)$ holds $v(t)(i + 1) = 1$. A traversal of the main loop in the program takes only finitely many steps, so we may assume that an execution of a step of it is a full traversal of the main loop.
We have to modify the program: At the beginning of the main loop we insert a block that checks if a limit time is reached. If the flag that corresponds to the maximal i such that $c_i \neq 0$, we decrease c_i by one. If $c_0 \neq 0$ we can't use the technique explained above. But counting finitely many steps is no problem. So the program checks at any limit time if ω^i additional steps are made. Since we do this beginning with the biggest we can count η steps in this way.
Let $a \in x$ code the pair $((\bar{I}, \bar{R}, \gamma), (\bar{I}', \bar{R}', \gamma))$. In the following code let `_I` denote $\bar{I}$, `_I'` denote $\bar{I}'$, `_R_{i}` denote $\bar{R}(i)$ and so forth. Let Z and Z' be to counters as described above. We may assume that they are correctly initialized from the input at the beginning of the program.
We replace every instruction P_i of P with the following block:

```
% Check if the configuration (_I, _R) is reached
if _I = i and _R_0 = R(0) and ... and _R_{r-1} = R(r-1)
then
  execute a step of Z
  if Z = _gamma then
    if result != 2 then result = 1
    reached_1 = 1
  end if
end if

% Check if the configuration (_I', _R') is reached
if _I' = i and _R'_0 = R(0) and ... and _R'_{r-1} =
R(r-1) then
  execute a step of Z'
  if Z = _gamma' return 0
    if result != 1 then result = 2
    reached_2 = 1
  end if
end if

P_i
```

As in the proof of the Generalized Speed-up Lemma we have to set up all these things carefully, but the techniques used there are also applicable in this case.

At the beginning the program has to check if the input is a valid coding of an element of x, whether it is a pair of two distinct elements of S and return 0 if this is not the case. We also insert a block at the end of the program, that returns 0 in the case that `reached_1` or `reached_2` is `0`, in which case one of the triples isn't in s. If this is not the case, then we return `1` if `result` is `1` and return `0` if not. Furthermore all jumps in P that terminate the program should be replaced with jumps to this block.

It remains to check if this program computes x. If the input isn't a code for a valid element of $\prec$, then we return 0. But if this is the case we count the occurrence of $(\bar{I}, \bar{R})$ and $(\bar{I}', \bar{R}')$, and decide $<$. So this program computes x. □

6 The lost melody theorem

Intuitively, the lost melody theorem says that there are reals that can be recognized, but not computed. In [2], this is shown in the context of ITTMs. Here, we prove the same holds for ITRMs. Accordingly, let us call a real r *recognizable* if the set of reals $\{r\}$ is computable in the empty oracle $\emptyset$.

Theorem 8 *There is a real r which is recognizable, but not computable. Thus, the Lost Melody Theorem holds for ITRM's as well.*

The rest of the paper is devoted to the proof of this theorem.

We need some notions from the fine structure theory of the constructible universe L.

$$\begin{aligned}
F_1(x, y) &= \{x, y\}\\
F_2(x, y) &= x \times y\\
F_3(x, y) &= \{(u, v){:}u \in x \wedge v \in y \wedge u \in v\}\\
F_4(x, y) &= x - y\\
F_5(x, y) &= x \cap y\\
F_6(x, y) &= \cup x\\
F_7(x, y) &= dom(x)\\
F_8(x, y) &= \{(u, v){:}(v, u) \in x\}\\
F_9(x, y) &= \{(u, v, w){:}(u, w, v) \in x\}\\
F_{10}(x, y) &= \{(u, v, w){:}(v, w, u) \in x\}
\end{aligned}$$

From now on, we define α to be the smallest ordinal such that $J_\alpha \models ZF^-$, where ZF^- is Zermelo–Fraenkel set theory without the power set axiom. Thus, α is a countable ordinal and J_α is itself countable. In fact, we have:

Lemma 5 *There is $s \in J_{\alpha+2}$ such that $s : \omega \to J_\alpha$ is surjective.*

Proof Let M_α be the Σ_ω Skolem hull of $\{J_\alpha\}$ in $J_{\alpha+1}$. All elements of M_α are of the form $h(i, \{J_\alpha\})$, where h is the canonical Σ_1 Skolem function for $J_{\alpha+1}$ (which is Σ_1 over $J_{\alpha+1}$ and hence an element of $J_{\alpha+2}$). Also, we have $M_\alpha \in J_{\alpha+2}$. Let $\pi_\beta(x){:}x \to y$ be the collapsing map for elements x of S_β, where y is the transitive collapse of x; this can be defined by induction on β and is easily seen to be an element of J_α for $\omega\alpha > \beta$. Furthermore, let π be the collapsing map for M_α, i.e. $\cup_\beta \pi_\beta$.

By the condensation lemma, the transitive collapse of M_α is of the form J_γ for some ordinal $\gamma \leq \alpha + 1$. Since $J_{\alpha+1} \models$ 'there is a maximal J-stage' (namely J_α), the same holds in J_γ and so $\gamma = \delta + 1$ for some ordinal δ. Furthermore, for each axiom ϕ of ZF^-, $J_{\alpha+1} \models \phi^{J_\alpha}$, and hence $J_\gamma \models \phi^{J_\delta}$. Now, since α was minimal, it follows that $\alpha \leq \delta$, so we get $\alpha = \delta$. Now $f{:}i \to \pi(h(i, \{J_\alpha\}))$ is a partial surjection from ω onto $J_{\alpha+1}$. Define $s(x) = f(x)$ if $f(x) \in J_\alpha$, $\emptyset$ otherwise. Since J_α and $J_{\alpha+1}$ are in $J_{\alpha+2}$ and the latter is closed under rudimentary functions, f is easily seen to be an element of $J_{\alpha+2}$ as well and is the desired surjection.

Let $p{:}\omega \times \omega \to \omega$ be the Cantor pairing function. Given a surjective map s as above, we can code J_α by a real r in a canonical way by simply putting $n = p(i, j)$ into r iff $s(i) \in s(j)$. Conversely, any real can be interpreted as a (possibly ill-founded) countable $\in$-structure in this way: Introduce countably many constants c_i and let $c_i E c_j \leftrightarrow p(i, j) \in r$. We say that r codes a model of ZF^- iff the $\in$-structure obtained in this way is such a model. (Obviously, any structure obtained in this way is transitive.) From now on, r denotes the $<_L$-minimal real that codes a $J_\alpha \models ZF^-$. Since a real coding J_α is easily generated from s as in Lemma 8 by applying some Gödel functions, we have $r \in J_{\alpha+2}$. From now on, we write $P^\emptyset(n)$ for the output that the program P generates from the input n in the empty oracle.

We start by proving:

Lemma 6 *r is not computable.*

Proof Suppose for the sake of a contradiction that P computes r. Since computations are absolute between transitive models of ZF$^-$, there is an $\in$-formula $\phi(v)$ such that $P^\emptyset(n) = 1 \leftrightarrow J_\alpha \models \phi(n)$. Since comprehension holds in J_α, we have $r \in J_\alpha$. But then, since J_α satisfies replacement, the structure coded by r is itself an element of J_α, and we get $J_\alpha \in J_\alpha$, a contradiction. □

The algorithm for deciding whether or not the oracle number o is equal to r proceeds in three steps: First, it is checked whether the $\in$-structure R coded by o (in the sense mentioned above) is well-founded. This can be done as in Sect. 3. If it doesn't succeed, we stop with negative result. If it does, we have to check whether all axioms of ZF$^-$ + V = L are valid in R and R is $\in$-minimal with this property. How to do this will follow easily from the effort taken for the last step: Assuming that the last step was successful (so o codes an $\in$-minimal model of ZF$^-$ + V = L), we have to check whether o is $<_L$-minimal with this property. For this purpose, we fix the oracle number o for the rest of the proof.

Since it is checked by now that R is isomorphic to a transitive, well-founded, $\in$-minimal model of ZF$^-$ + V = L, we may assume that R is of the form J_γ for some ordinal γ.

The J-hierarchy is obtained by iterating the process of closing $J_\beta \cup \{J_\beta\}$ under Gödel functions and taking unions at limits. Each element of $L = \bigcup_\beta J_\beta$ can therefore be represented as an iteration of Gödel functions applied to several of the S_γ. We view such a representation as a name for the element; if we restrict ourselves to an initial segment of L below a countable ordinal, this concept can be arithmetized, which will allow us to decide $\in$-formulas relativized to $J_{\alpha+2}$ when a J_α-oracle is given, where J_α is the minimal ZF$^-$-model as above.

We start by assigning natural numbers to the constituting elements of names; having a surjection s as in the introduction at our disposal, we let $3n$ code $s(n)$. $S_{\omega(\alpha+i)+j}$, $j \in \omega$, $i \in \{0, 1\}$ is represented by $3j + i + 1$. Names can now be coded by a suitable application of the pairing function p:

Definition 7 A *name* is any number generated in the following way:

(i) $p(2n, i)$ is a name for all $i, \quad n \in \omega$
(ii) if a and b are names, $i \in \omega$, then so is $p(2i + 1, p(a, b))$.

Thus a name is an ordered pair $\langle a, b\rangle$ of naturals; the parity of the first element shows whether the name is flat, i.e., an S_β or an element of J_γ if a is even or whether and which Gödel function was applied. We explain the coding by giving the interpretation function I:

Definition 8 The interpretation function I is defined as follows:

(i) if $i = 3k + j$, $j \in \{1, 2\}$, then $I(p(2n, i)) = S_{\omega(\gamma+j-1)+k}$
(ii) otherwise, $I(p(2n, 3i)) = s(i)$
(iii) if $j = 10k + l$, $0 \le l < 10$, then $I(p(2j + 1, p(a, b))) = F_{l+1}(I(a), I(b))$

Obviously, we assign multiple (and in fact infinitely many) names to each interpretation. However, this has a technically advantageous consequence:

Proposition 3 *Every natural number is a name.*

This will allow us to search through $J_{\gamma+2}$ by searching through ω without any further checks.

The idea of a final constituent of a name is given by the following formal notion:

Definition 9 The argument set $A(n)$ of a name n is given by the following recursive rules:

(i) $A(p(2n, i)) = \{i\}$
(ii) $A(p(2k + 1, p(a, b))) = A(a) \cup A(b)$

The following is the main tool for inductive arguments and definitions on names. For rational q, $\lceil q\rceil$ denotes the smallest integer n such that $n \ge q$.

Definition 10 Let a be a name. Then $\text{ps}(a)$, the *pseudostage* of a, is defined as follows:

(i) for $i = 3k + j$, $j \in \{1, 2\}$, $\text{ps}(p(2n, i)) = \omega(\gamma + j - 1) + 3k$
(ii) otherwise, $\text{ps}(p(2n, i)) = 0$
(iii) if $A(a), A(b) \subseteq \{3i | i \in \omega\}$, then $\text{ps}(p(2k+1, p(a, b)))=\max\{ps(a), ps(b)\}+1$
(iv) if $k = 10n + 1$ for some $n \in \omega$ $\text{ps}(p(2k + 1, p(a, b))) = \max\{\text{ps}(a), \text{ps}(b)\} + 1$
(v) for $k = 10n + j$, $n \in \omega$, $j \in \{4, 5, 6\}$, if $\max\{\text{ps}(a), \text{ps}(b)\} = \omega(\gamma + j) + t$, let $\text{ps}(p(2k + 1, p(a, b))) = \omega(\gamma + j) + 3\lceil \frac{t}{3}\rceil + 1$
(vi) for $k = 10n+j$, $n \in \omega$, $j \in \{2, 3, 7, 8, 9, 0\}$, if $\max\{\text{ps}(a), \text{ps}(b)\} = \omega(\gamma+j)+t$, let $\text{ps}(p(2k + 1, p(a, b))) = \omega(\gamma + j) + 3\lceil \frac{t}{3}\rceil + 2$

We call a name m *minimal* if any name n with $I(n) = I(m)$ satisfies $\text{ps}(n) \ge \text{ps}(m)$.

In our arithmetization, the ordinal $\omega(\gamma + j) + i$ will be coded by $p(j, i)$. Accordingly, we slightly abuse our notation by viewing ps as a function taking naturals to naturals rather than to ordinals. If we talk about relations between pseudostages like $<$, we nevertheless mean the ordinals, and similarly for $\text{ps}(a)+2$ etc. Since the definition consists of easy recursive rules, which can be implemented even on a classical (finite) register machine, we note:

Proposition 4 *The pseudostage of a name can be computed by an ITRM-program in finite time.*

From now on, if a and b are names, we write $a\tilde{\in}b$ and $a\tilde{=}b$ instead of $I(a) \in I(b)$ and $I(a) = I(b)$. Furthermore, we write $a <_{\text{ps}} b$ for $\text{ps}(a) < \text{ps}(b)$, similarly for $>, =$ etc. If β is an ordinal $a <_{\text{ps}} \beta$ means $\text{ps}(a) < \beta$. Sometimes we will write $a <_{\text{ps}+i} b, i \in \omega$ to indicate that $\text{ps}(a) + i < \text{ps}(b)$.

The following lemma is the main reason for the usefulness of the pseudostage. To enhance readability, we will e.g. write $\langle 8, x, y\rangle$ instead of $p(8, p(x, y))$.

Lemma 7 *Suppose a and b are names such that $I(a) \in I(b)$. Then*:

(i) *If $ps(b) > 0$ then there is a name c such that $ps(c) < ps(b)$ and $I(c) = I(a)$. Thus, minimal names of elements of sets with names of $ps > 0$ have a strictly smaller pseudostage.*
(ii) *If $ps(b) = 0$ then there is a name c such that $ps(c) = 0$ and $I(c) = I(a)$.*

Proof (a) Easy induction on the pseudostage. To give a feeling for the kind of argument used here, we prove this for names of the form $\langle 8, x, y\rangle$. In the following, all names are chosen minimal. Consider $z \in \langle 8, x, y\rangle$, so that z is of the form $\langle v, u\rangle$, where $\langle u, v\rangle \in x$. By definition of ps, we have $x <_{\text{ps}+1} \langle 8, x, y\rangle$; now, by induction, $\langle u, v\rangle <_{\text{ps}} x, \{u, v\} <_{\text{ps}} \langle u, v\rangle, u <_{\text{ps}} \{u, v\}, v <_{\text{ps}} \{u, v\}$. Since pairing (i.e. application of F_1) increases the pseudostage by 1, we have $\langle v, u\rangle <_{\text{ps}} x <_{\text{ps}} \langle 8, x, y\rangle$. (b) By transitivity of J_γ. □

We will now define $a \in b$ and $a = b$ by induction on a partial order $\lhd$ of the triples $\langle \sigma, a, b\rangle$, where $\sigma \in \{\in, =\}$, a, b names without using the interpretation function. This will allow a purely syntactical decision procedure for atomic formulas by inspection of the names.

Definition 11 For triples as mentioned above, let $m_a = \max\{\text{ps}(a_1), \text{ps}(a_2)\}$, $m_b = \max\{\text{ps}(b_1), \text{ps}(b_2)\}$. Then define $\langle \sigma_1, a_1, a_2\rangle \lhd \langle \sigma_2, b_1, b_2\rangle$ iff one of the following holds:

(i) $m_a < m_b$
(ii) $m_a = m_b$, and left triple satisfies that σ_1 is $\in$ and $a_1 <_{ps} a_2$, while the analogous proposition for the right triple is not true
(iii) $m_a = m_b$ and the left, but not the right triple satisfies that $\sigma_1$is $=$.

Thus, given that we already know what $\in$ and $=$ mean for names with ps $< \beta$, we first explain $a \in b$ for $\text{ps}(a) < \beta$. Now, since elements of sets have names of smaller pseudostages than the sets themselves, we can define $a = b$ for names with ps $\leq \beta$ and then also $a \in b$ for $\beta = \text{ps}(a) > \text{ps}(b)$, since this is only possible if there is a name c with $\text{ps}(c) < \beta$ and $c = a$.

This approach leads to a meaningful definition: Since the maximum of the pseudostages cannot increase when going down in $\lhd$, and since we can go down at most two steps while preserving the maximum and these maxima are ordinals, we have the following:

Proposition 5 $\lhd$ *is well-founded.*

We will now give a formal version of the above sketch by induction on $\lhd$.

Definition 12 For the sake of brevity, we abbreviate names and write e.g. $a\tilde{\in}\{x, y\}$, where we really mean $a\tilde{\in}\langle 1, x, y\rangle$, and similar for ordered pairs and triples. The replacement function η assigns (codes of) formulas to (codes of) formulas as follows:

(i) $a\tilde{=}b \longmapsto \forall x <_{ps} \max\{1, \mathrm{ps}(a), \mathrm{ps}(b)\}(x\tilde{\in}a \leftrightarrow x\tilde{\in}b)$
(ii) if $\mathrm{ps}(a) = \mathrm{ps}(b) = 0$, then $(a\tilde{\in}b) \longmapsto$ true, if $p(a, b) \in o$, false, otherwise
(iii) if $\mathrm{ps}(a) > 0, \mathrm{ps}(a), \mathrm{ps}(b) \in \omega$, then $(a\tilde{\in}b) \longmapsto (\exists a_0, b_0 =_{\mathrm{ps}} 0(a_0\tilde{=}a \wedge b_0\tilde{=}b \wedge a_0\tilde{\in}b_0))$

If $\mathrm{ps}(b) = \beta \notin \omega, \mathrm{ps}(a) < \beta$:

(i) $(a\tilde{\in}b = \langle 1, x, y\rangle) \longmapsto (a\tilde{=}x \vee a\tilde{=}y)$
(ii) $(a\tilde{\in}b = \langle 2, x, y\rangle) \longmapsto (\exists t_1 <_{ps} x\exists t_2 <_{ps} y(t_1\tilde{\in}x \wedge t_2\tilde{\in}y \wedge \langle t_1, t_2\rangle\tilde{=}a))$
(iii) $(a\tilde{\in}b = \langle 3, x, y\rangle) \longmapsto (\exists t_1 <_{ps} x\exists t_2 <_{ps} y(t_1\tilde{\in}x \wedge t_2\tilde{\in}y \wedge \langle t_1, t_2\rangle\tilde{=}a \wedge t_1\tilde{\in}t_2))$
(iv) $a\tilde{\in}b = \langle 4, x, y\rangle \longmapsto a\tilde{\in}x \wedge a\tilde{\notin}y$
(v) $a\tilde{\in}b = \langle 5, x, y\rangle \longmapsto a\tilde{\in}x \wedge a\tilde{\in}y$
(vi) $a\tilde{\in}b = \langle 6, x, y\rangle \longmapsto \exists z <_{\mathrm{ps}} x(z\tilde{\in}x \wedge a\tilde{\in}z)$
(vii) $a\tilde{\in}b = \langle 7, x, y\rangle \longmapsto \exists u, v <_{\mathrm{ps}+3} x\exists z <_{\mathrm{ps}} x(z\tilde{=}\langle u, v\rangle \wedge a\tilde{=}u)$
(viii) $a\tilde{\in}b = \langle 8, x, y\rangle \longmapsto \exists z <_{\mathrm{ps}} x\exists u, v <_{\mathrm{ps}+1} z(z\tilde{=}\langle u, v\rangle \wedge a\tilde{=}\langle v, u\rangle \wedge z\tilde{\in}x)$
(ix) $a\tilde{\in}b = \langle 9, x, y\rangle \longmapsto \exists z <_{\mathrm{ps}} x\exists u <_{\mathrm{ps}+1} z\exists v, w <_{\mathrm{ps}+3} z(z\tilde{=}\langle u, w, v\rangle \wedge z\tilde{\in}x \wedge a\tilde{=}\langle u, v, w\rangle)$
(x) $a\tilde{\in}b = \langle 10, x, y\rangle \longmapsto \exists z <_{\mathrm{ps}} x\exists v <_{\mathrm{ps}+1} z\exists w, u <_{\mathrm{ps}+3} z(z\tilde{=}\langle v, w, u\rangle \wedge z\tilde{\in}x \wedge a\tilde{=}\langle u, v, w\rangle)$
(xi) b is an S-stage: $j \in \{1, 2\}, a\tilde{\in}b = p(2n, 3k + j) \longmapsto \exists c \leq_{\mathrm{ps}} a(c\tilde{=}a \wedge c <_{\mathrm{ps}} b)$

If $\mathrm{ps}(\alpha) \geq \mathrm{ps}(\beta)$:
$a\tilde{\in}b \longmapsto \exists c <_{\mathrm{ps}} b(a\tilde{=}c \wedge c\tilde{\in}b)$

If $0 = \mathrm{ps}(a) < \mathrm{ps}(b) \in \omega$ then the above almost works. Just replace each $<_{\mathrm{ps}} x$ by $<_{\mathrm{ps}} \max\{1, \mathrm{ps}(x)\}$ and terms like $\langle u, v\rangle$ by their definition (so if, for example, $\{u, v\}$ appears, replace it by a new variable c and add the condition $\forall x <_{\mathrm{ps}} \max\{1, \mathrm{ps}(c)\}(x \in c \leftrightarrow x = u \vee x = v)$; similarly for ordered pairs and triples.)

This function produces for every triple $\langle s, x, y\rangle$, where s is $\tilde{\in}$ or $\tilde{=}$, an equivalent formula which is only based on $\lhd$-smaller atomic formulas. This procedure can be implemented on an ITRM.

For this, an arithmetization of the appearing formulas is needed: So set $a(x\tilde{\in}y) = 5p(x, y), a(x\tilde{=}y) = 5p(x, y) + 1, a(\phi \wedge \psi) = 5p(a(\phi), a(\psi)) + 2, a(\neg\phi) = 5a(\phi) + 3, a(\exists t_i\psi) = 5p(p(i, a(\psi)) + 4$. A formula of the form $\exists t_i <_{\mathrm{ps}+j}\phi$ is viewed as $\exists t_i(\mathrm{ps}(t_i) + j < \mathrm{ps}(x) \wedge \phi$ in this respect; this will, in connection with the fact that the implementation considers conjunctions from left to right, lead to the termination of the algorithm. For the sake of uniformity, we introduce the symbol Ω and write unbounded quantifiers like $\exists x\phi$ as $\exists x <_{\mathrm{ps}} \Omega\phi$. We now describe a stack algorithm for deciding $\in$-formulas in $J_{\gamma+2}$.

The implementation essentially uses only two registers, one of which contains a sequence of (codes of) $\in$-formulas coded by iterating the pairing function, while the others holds a status for the most recently processed element of this sequence (true, false, unknown, represented by 0, 1, 2, respectively). In addition, numerous auxiliary registers are used for calculating the auxiliary functions. We leave out those details.

For the description, we use sequences of pairs of the form $\langle f_1, s_1\rangle \longmapsto \langle f_2, s_2\rangle$, where the first element represents the sequence of formulas, the second the status; the reader will easily convince himself that the described development of the stack contents can easily be generated by a standard register machine without assigning other values to the two central registers in between. $\langle\rangle$ is the empty sequence, $\langle S|e\rangle$, $S = \langle s_1, \ldots, s_n\rangle$ denotes the sequence $\langle s_1, \ldots, s_n, e\rangle$; $\phi[x/i]$ for $i \in \omega$ is the formula derived from ϕ by replacing every free occurrence of x in ϕ by i.

Base cases:

$\langle\langle\rangle, 1\rangle : output = true;$, $\langle\langle\rangle, 0\rangle : output = false$, $\langle\langle\rangle, ?\rangle : output = true$
$\langle\langle S|false\rangle, ?\rangle \longmapsto \langle S, 0\rangle$
$\langle\langle S|true\rangle, ?\rangle \longmapsto \langle S, 1\rangle$
Atomic formulas, $s \in \{\in, =\}$:
$\langle\langle S|s(x, y)\rangle, ?\rangle \longmapsto \langle\langle S|\eta(s(x, y))\rangle, ?\rangle$ (where η is the replacement function defined above; we assume that the formula on the right hand side is rewritten in such a way that in contains only $\exists$, $\neg$ and $\wedge$ as logical symbols.)
Conjunction:
$\langle\langle S|\phi \wedge \psi\rangle, ?\rangle \longmapsto \langle\langle\langle S|\phi \wedge \psi\rangle|\phi\rangle, ?\rangle$
$\langle\langle S|\phi \wedge \psi\rangle, 0\rangle \longmapsto \langle S, 0\rangle$
$\langle\langle S|\phi \wedge \psi\rangle, 1\rangle \longmapsto \langle\langle S|\psi\rangle, ?\rangle$
Negation:
$\langle\langle S|\neg\phi\rangle, ?\rangle \longmapsto \langle\langle\langle S|\neg\rangle|\psi\rangle, ?\rangle$
$\langle\langle S|\neg\rangle, 0\rangle \longmapsto \langle S, 1\rangle$
$\langle\langle S|\neg\rangle, 1\rangle \longmapsto \langle S, 0\rangle$
Existential quantifier:
$\langle\langle S|\exists x\phi\rangle, ?\rangle \longmapsto \langle\langle\langle S|\langle\exists x\phi, 0\rangle\rangle|\phi[x/0]\rangle, ?\rangle$
$\langle\langle S|\langle\exists x\phi, k\rangle\rangle, 1\rangle \longmapsto \langle S, 1\rangle$
$\langle\langle S|\langle\exists x\phi, k\rangle\rangle, 0\rangle \longmapsto \langle S, 0\rangle \longmapsto$
$\langle\langle\langle S|\langle\exists x\phi, k+1\rangle\rangle|\phi[x/k+1]\rangle, ?\rangle$

We will now show that this algorithm, given the input $\langle\langle\phi\rangle, ?\rangle$, ϕ an $\in$-formula without free variables, always terminates and returns the truth value of $J_{\alpha+2} \models \phi$. We do this by induction on a well-order on these formulas.

In the following, $\text{at}(\phi)$ is the set of atomic subformulas of ϕ, written in the form $\langle\in, x, y\rangle$ etc. First, write ϕ in prenex normal form and bound all unbounded quantifiers with the help of the symbol Ω as introduced above.

For $\beta = \omega\gamma + j$, we set $\beta - i = \omega\gamma + (j - i)$ for $i \leq j$ and otherwise $\beta - i = \omega\gamma$.

Definition 13 For such a formula ψ we define $\text{pt}(\psi)$, the potential of ψ, as follows:

(i) $\text{pt}(\exists x <_{\text{ps}+i} y\phi) = \lhd - \max\{\psi[x/\text{ps}(y) - i]|\psi \in \text{at}(\phi)\}$
(ii) $\text{pt}(\neg\phi) = \text{pt}(\phi)$
(iii) $\text{pt}(\phi \wedge \psi) = \lhd - \max\{\text{pt}(\phi), \text{pt}(\psi)\}$

Intuitively, $\text{pt}(\psi)$ is an upper bound for the complexity of an atomic formula that has to be decided in order to evaluate ψ. For our purposes, a slightly finer order is necessary:

Definition 14 If ϕ and ψ are formulas as described above, we let $\phi <_{\mathrm{F}} \psi$ iff one of the following cases occurs:

(i) $\mathrm{pt}(\phi) \lhd \mathrm{pt}(\psi)$
(ii) $\mathrm{pt}(\phi)$ and $\mathrm{pt}(\psi)$ are incomparable in $\lhd$ and ϕ is a proper subformula of ψ.

Proposition 6 *$<_F$ is a well-order on formulas of this kind.*

By case distinction and the definition of the replacement function:

Lemma 8 *Whenever the algorithm puts a new formula ϕ on the stack on top of the formula ψ, we have $\phi <_F \psi$.*

Therefore, finally:

Lemma 9 *The algorithm terminates and gives the correct result.*

Proof By induction on $<_{\mathrm{F}}$ with the help of the last lemma and the last proposition; observe that the bounding of a quantifier is always processed as the first conjunct by the way the algorithm treats conjunctions and that the complexity drop is therefore mirrored by the processing steps. The only interesting case is existential quantification: If $\exists x\phi$ is true, a witnessing x will be found, the formula will be taken off the stack, and the status register will be set to 1. If it is false, the seemingly pointless step in the last line in the description of the algorithm forces the occurrence of a limit state, in which the formula is off the stack and the status register contains a 0. □

Thus, we are now able to decide arbitrary $\in$-formulas in $J_{\alpha+2}$.

Finally, we have to check the $<_L$-minimality of o. Since in this case, we have $o = r$ and we know that $r \in J_{\alpha+2}$, we can do this by finding a name n for o and then checking for each name u whether $I(u) <_L r$ and u codes an $\in -minimal$ model of ZF^-. We just gave a procedure for the latter; the well-order $<_L$ of the constructible hierarchy (restricted to $J_{\alpha+2}$) can be expressed by an $\in$-formula in $J_{\alpha+2}$ and thus computed by the same method. Since there are only countably many names, we will have a way to test for $<_L$-minimality of a real given in the oracle as soon as we can tell how to find n such that $I(n) = o$. Again, since the number of names is countable, it suffices to be able to test for some given name m and some oracle number z whether or not $I(m) = z$.

For this, we first run through all the names until we find one, say y_0, such that $\neg\exists t(t \in I(y_0))$, that is, $I(y_0) = \emptyset$ and save it in a separate register.

Definition 15 For $k \in \omega$ the canonical name $\mathrm{cn}(k)$ of k is defined as follows:

(1) $\mathrm{cn}(0) = y_0$
(2) $\mathrm{cn}(k+1) = \langle 6, \langle 1, \mathrm{cn}(k), \langle 1, \mathrm{cn}(k), \mathrm{cn}(k)\rangle\rangle, 0\rangle$

Proposition 7 *$I(cn(k)) = k$ for $k \in \omega$*

Proof By definition of y_0 for $k = 0$, else $I(\mathrm{cn}(k+1))$ is just
$\cup\{I(\mathrm{cn}(k)), \{I(\mathrm{cn}(k)), I(\mathrm{cn}(k))\}\}$, which, by induction, equals
$\cup\{k, \{k, k\}\} = \cup\{k, \{k\}\} = k \cup \{k\} = k + 1$. □

$\mathrm{cn}(k)$ is obviously easy to compute. So we can check for a name m whether $I(m) \in \omega$ simply by checking for any $i \in \omega$ whether $I(m) = I(\mathrm{cn}(i))$, at the same time finding the corresponding i in case of success. From this, one constructs an algorithm for checking $I(m) \subset \omega$ by running through the names and testing for being an element of $I(m)$ and of ω.

To find out if $z \subseteq I(m)$, run through ω, checking by oracle call for every $k \in \omega$ whether $k \in z$, then, if so, whether $c(k) \in I(m)$.

Finally, check $I(m) \subseteq z$ by finding out if $I(m) \subseteq \omega$ and, if so, running through the canonical names of all $k \in \omega$ and calling the oracle for each $\mathrm{cn}(k) \in m$ to see if $k \in z$.

This concludes the description of the algorithm, and thus the proof of the lost melody theorem.

References

1. Dimitriou, I., Hamkins, J.D., Koepke, P. (eds.): BIWOC—Bonn International Workshop on Ordinal Computability, Bonn Logic Reports (2007)
2. Hamkins, J.D., Lewis, A.: Infinite time turing machines. J. Symb. Log. **65**(2), 567–604 (2000)
3. Hamkins, J.D., Linetsky, D., Miller, R.: The complexity of quickly ORM-decidable sets. In: Cooper, S.B., et al. (eds.) Computation and Logic in the Real World, LNCS, vol. 4497, pp. 488–496. Springer, Heidelberg (2007)
4. Hamkins, J.D., Miller, R.: Post's problem for ordinal register machines. In: Cooper, S.B., et al. (eds.) Computation and Logic in the Real World, LNCS. vol. 4497, pp. 358–367. Springer, Heidelberg (2007)
5. Koepke, P., et al.: Infinite time register machines. In: Beckmann, A., (ed.) Logical Approaches to Computational Barriers, LNCS. vol. 3988, pp. 257–266. Springer, Heidelberg (2006)
6. Koepke, P.: Turing computations on ordinals. B. Symb. Log. **11**, 377–397 (2005)
7. Koepke, P., Miller, R.: An enhanced theory of infinite time register machines. In: Beckmann, A., et al. (eds.) Logic and Theory of Algorithms LNCS, vol. 5028, pp. 306–315. (2008)
8. Koepke, P., Siders, R.: Register computations on ordinals. Arch. Math. Log. **47**, 529–548 (2008)
9. Koepke, P., Seyfferth, B.: Ordinal machines and admissible recursion theory. Ann. Pure Appl. Log. **160**, 310–318 (2009)
10. Koepke, P., Siders, R.: Computing the recursive truth predicate on ordinal register machines. In: Beckmann, A., et al. (eds.) Logical Approaches to Computational Barriers, Computer Science Report Series, vol. 7, pp. 160–169. (2006)
11. Koepke, P., Siders, R.: Minimality considerations for ordinal computers modeling constructibility. Theoretical Comput. Sci. **394**, 197–207 (2008)
12. Jech, T.: Set Theory, 3rd Millennium edition, revisited and expanded. Springer, Berlin (2002)
13. Jensen, R.B.: The fine structure of the constructible hierarchy. Ann. Math. Log. (1972)