Post's Problem for Ordinal Register Machines

Joel D. Hamkins[1] and Russell G. Miller[2] *

[1] Department of Mathematics, College of Staten Island, and Doctoral Program in Mathematics, The CUNY Graduate Center
[2] Department of Mathematics, Queens College, and Doctoral Program in Computer Science, The CUNY Graduate Center

jhamkins@gc.cuny.edu
Russell.Miller@qc.cuny.edu

Abstract. We study Post's Problem for the ordinal register machines defined in [6], showing that its general solution is positive, but that any set of ordinals solving it must be unbounded in the writable ordinals. This mirrors the results in [3] for infinite-time Turing machines, and also provides insight into the different methods required for register machines and Turing machines in infinite time.

Keywords: computability, ordinal computability, ordinal register machine, Post's Problem.

1 Definitions

Ordinal register machines, or ORM's, are defined and described by Koepke and Siders in [6]. They generalize the traditional finite-time register machines: the registers are now allowed to contain any ordinal value, not just natural numbers, and the program runs through ordinal time, with its state at limit-ordinal stages determined in a natural way by taking liminf's of the cells and states at the preceding stages. Koepke and Siders proved that the sets of ordinals computable by an ORM, with finitely many ordinal parameters, are precisely the constructible sets of ordinals, i.e. those lying in Gödel's constructible universe L.

Since the ordinal register machine programs are finite, they each refer to only finitely many registers, and the memory used by any ORM algorithm is correspondingly limited to these fixed finite number of ordinal values at any time. This contrasts with the situation for the infinite time Turing machines of Hamkins and Lewis [2] and for the ordinal Turing machines of Koepke [5], where the algorithms can store information stretching out on an transfinite tape.

The original version of Post's Problem applied to finite-time Turing machines. It asked whether there exists a computably enumerable set A which is neither

* The authors were partially supported by several grants from the Research Foundation of CUNY. In addition, they are thankful for support provided by the Hausdorff Center in Mathematics at the University of Bonn in connection with the Bonn International Workshop on Ordinal Computability 2007.

computable nor complete. That is, it required $\emptyset <_T A <_T \emptyset'$, where $\emptyset'$ is the jump of the empty set, or equivalently the Halting Problem for finite-time Turing machines. Post's Program for solving this problem was to discover a nonvacuous property of c.e. sets, expressible using only the containment relation $\subseteq$, which would guarantee that A was incomplete and noncomputable.

Post's Program was the genesis of the notions of simple, hypersimple, and hyperhypersimple sets, all of which properties Post originally hoped would fulfill his program. In fact, none of these properties implies incompleteness, and Post did not live to see the solution of the problem that bears his name. Post's Program was completed by Harrington and Soare [4] in 1991, but his Problem was solved much earlier, in 1956 and 1957, with the invention of the finite injury priority method (independently) by Friedberg [1] and Muchnik [8]. A good description of this method appears in section VII.2 of [10].

The notion of a computably enumerable subset of ω extends naturally to our context: a set of ordinals is *ORM-enumerable*, or *semidecidable*, if it is the domain (equivalently, the range) of some ORM. Shortly we will also define the jump operation for ORM's. Then we will ask the analogue of Post's Problem for sets of ordinals under computation by ORM's.

An ordinal α is ORM-*writable* if there is an ordinal register machine which, on input 0, halts and outputs α. Briefly, $\varphi_e(0)\downarrow = \alpha$ for some $e \in \omega$. Also, an ordinal σ is ORM-*clockable* if some computation $\varphi_e(0)$ halts after exactly σ steps. Notice that it is ORM-computable whether σ is clockable, since we can run all computations $\varphi_e(0)$ for σ-many steps and check whether any of them halted after exactly σ steps. On the other hand, the set of writable ordinals is ORM-enumerable, but not ORM-computable, essentially because a computation which halts after a very long number of steps could still have a relatively small ordinal as its output.

We write $\varphi_{e,\sigma}(\alpha)\downarrow$ to signify that the program φ_e converges on input α in *fewer than* σ steps. This notation is different from the common usage in finite-time computability theory, where $\varphi_{e,s}(n)\downarrow$ denotes convergence in $\leq s$ steps; our way is more appropriate in a context where we must deal with limit ordinals.

The *weak jump* $\emptyset^{\Diamond}$ of the empty set is the set

$$\emptyset^{\Diamond} = \{e \in \omega : \varphi_e(0)\downarrow\}.$$

We have approximations to the weak jump:

$$\emptyset^{\Diamond}_{\sigma} = \{e : \varphi_{e,\sigma}(0)\downarrow\};$$

these are nested upwards, and are computable uniformly in σ. Also notice that σ is clockable iff $\emptyset^{\Diamond}_{\sigma} \neq \emptyset^{\Diamond}_{\sigma+1}$, and that for limit ordinals λ, $\emptyset^{\Diamond}_{\lambda} = \cup_{\sigma<\lambda}\emptyset^{\Diamond}_{\sigma}$. (This would fail if we had kept the finite-time notation for $\varphi_{e,\lambda}(\alpha)\downarrow$.) We also have the *strong jump* $\emptyset^{\blacklozenge}$ of $\emptyset$ and its computable approximations $\emptyset^{\blacklozenge}_{\sigma}$:

$$\emptyset^{\blacklozenge} = \{\langle e, \alpha\rangle : \varphi_e(\alpha)\downarrow\} \subset \omega \times \mathrm{ON} \qquad \emptyset^{\blacklozenge}_{\sigma} = \{\langle e, \alpha\rangle : \varphi_{e,\sigma}(\alpha)\downarrow\}.$$

The strong jump is the actual halting problem in ORM-computability; the weak jump, roughly analogous to the jump in finite-time computability, is just the

most convenient way to diagonalize and build a noncomputable set. In finite time, of course, the jump and the halting problem are computably isomorphic, but in our context this is no longer true; indeed $\emptyset^\diamondsuit <_{ORM} \emptyset^\blacklozenge$, under the standard definition of oracle computability for ordinal register machines.

The version of the following lemma for infinite time Turing machines was a significant result, proved by Philip Welch in [11], but in the ordinal register machine context we may observe it easily.

Lemma 1. *For ordinal register machines, the supremum γ of the clockable ordinals equals the supremum λ of the writable ordinals.*

Proof. Every clockable ordinal α is writable: just run the computation $\varphi_p(0)$ which halts after α steps, adjusting the machine so that at each step, it increments the ordinal in a new step register. When $\varphi_p(0)$ halts, transfer the contents of the step register to the output register and then halt. Thus $\gamma \leq \lambda$. Conversely, if $\alpha = \varphi_q(0)$ is writable, then $\varphi_q(0)$ takes at least α-many steps to halt, since after β steps, no register can contain any ordinal $> \beta$. Hence $\lambda \leq \gamma$. □

2 Post's Problem

Now we begin to consider Post's Problem. First we ask whether there exist relatively simple ORM-enumerable sets (subsets of ω, for instance) which are noncomputable and incomplete. The answer is no.

Theorem 1. *No subset $C \subseteq \omega$ satisfies $\emptyset <_{ORM} C <_{ORM} \emptyset^\diamondsuit$. Indeed, the same holds for subsets $C \subseteq \rho$, for any writable ordinal ρ.*

Proof. Consider a set $C \subseteq \rho$ with $\emptyset \leq_{ORM} C \leq_{ORM} \emptyset^\diamondsuit$ and ρ writable. This part of the proof is similar to that of Theorem 2.1 in [3], the corresponding result for infinite-time Turing machines.

Recall that $\emptyset^\diamondsuit_\sigma = \{e \in \omega : \varphi_{e,\sigma}(0)\downarrow\}$. This is a computable enumeration of the semidecidable set $\emptyset^\diamondsuit$. By assumption there is a program q such that $\varphi_q^{\emptyset^\diamondsuit}$ computes the characteristic function of C. This gives us a computable approximation to C:

$$C_\sigma = \{\beta < \rho : (\exists \delta \geq \sigma)[\varphi^{\emptyset^\diamondsuit_\sigma}_{q,\delta}(\beta)\downarrow = 1 \ \&\ (\forall \theta)[\sigma < \theta \leq \delta \implies \emptyset^\diamondsuit_\sigma = \emptyset^\diamondsuit_\theta]]\}.$$

Since $\varphi_q^{\emptyset^\diamondsuit}$ is the characteristic function of C, it must be total. Indeed, since $\emptyset^\diamondsuit_\gamma = \emptyset^\diamondsuit$, we must have $C_\sigma = C$ for all $\sigma \geq \gamma$. Therefore we can compute, uniformly in β and σ, whether $\beta \in C_\sigma$: having written ρ and checked that $\beta < \rho$, just run $\varphi_q^{\emptyset^\diamondsuit_\sigma}(\beta)$ until we reach a stage $\delta \geq \sigma$ such that either $\emptyset^\diamondsuit_\delta \neq \emptyset^\diamondsuit_\sigma$ (so $\beta \notin C_\sigma$) or $\varphi^{\emptyset^\diamondsuit_\sigma}_{q,\delta}(\beta)\downarrow$. If we find that this computation halts and outputs 1, before the approximation to $\emptyset^\diamondsuit$ changes, then $\beta \in C_\sigma$; if it outputs a different value, then $\beta \notin C_\sigma$. (In finite-time computability, the analogous process is the construction of a computable approximation to an arbitrary set $\leq_T \emptyset'$.)

Lemma 2. *With this approximation, if $C_\sigma \neq C_{\sigma+1}$, then $\sigma+1$ is clockable (and hence σ is writable).*

Proof. If $\emptyset^\diamondsuit_\sigma \neq \emptyset^\diamondsuit_{\sigma+1}$, then some computation $\varphi_e(0)$ converged in $(\sigma+1)$-many steps, so $(\sigma+1)$ is clockable. Otherwise, the definition of C_σ shows that $C_\sigma = C_{\sigma+1}$, as follows. To check whether some $\beta < \rho$ lies in C_σ, we find the least $\delta \geq \sigma$ for which either $\emptyset^\diamondsuit_\sigma \neq \emptyset^\diamondsuit_\delta$, or $\varphi^{\emptyset^\diamondsuit_\sigma}_{q,\delta}(\beta)\downarrow$. If $\delta \geq \sigma+1$, then we find the same δ when checking whether $\beta \in C_{\sigma+1}$, so the answer is the same. If $\delta = \sigma$, then $\varphi^{\emptyset^\diamondsuit_\sigma}_{q,\sigma}(\beta)\downarrow$, and hence $\varphi^{\emptyset^\diamondsuit_{\sigma+1}}_{q,\sigma+1}(\beta)\downarrow = \varphi^{\emptyset^\diamondsuit_\sigma}_{q,\sigma}(\beta)$ as well, since the oracle has not changed. □

We now consider two cases. First, if there exists some $\beta < \gamma(=\lambda)$ such that $C_\beta = C$, then C is ORM-computable. To compute C, we run some fixed program $\varphi_p(0)$ which writes the least writable ordinal $\delta \geq \beta$ and then computes C_δ. By Lemma 2, $C_\sigma = C_\beta = C$ for every σ with $\beta \leq \sigma < \delta$. But then $\emptyset^\diamondsuit_\delta = \emptyset^\diamondsuit_\beta$ as well, since $\emptyset^\diamondsuit_\delta$ contains those programs which halt *before* stage δ, and so the definition of C_σ shows that $C_\delta = C_\beta = C$. Hence we have computed C.

Otherwise there is no such β, and in this case $\emptyset^\diamondsuit \leq_{ORM} C$, since with a C-oracle we can search for the least σ such that $C_\sigma = C$. (Here we need to know that C is contained in ρ, as assumed by the theorem. We can write ρ, and then to verify that $C = C_\sigma$, we need only check that all $\alpha < \rho$ lie in C iff they lie in C_σ.) But by assumption, the σ we find is $\geq \gamma$ (in fact precisely γ), and when we find it, we write it on the output tape and halt. Thus γ is C-writable, and with γ it is easy to compute $\emptyset^\diamondsuit$. □

So Post's Problem has a negative solution when we restrict to sets C such that the supremum of C is writable. This is a large class of sets: it includes every non-cofinal subset of γ. However, without this restriction, the same problem has a positive solution. We prove this by a construction in the style of Friedberg and Muchnik. First we give a necessary lemma.

Lemma 3 (Reflection Lemma for ORM's). *Suppose that $\varphi^A_e(x)\downarrow = 0$, where x is a writable ordinal and A is a semidecidable set of ordinals, with ORM-computable enumeration $\langle A_\sigma \rangle$ such that $A_\gamma = A$. (This means that $A_\sigma \subseteq A_\tau$ for all $\sigma < \tau$, that $A_\beta = \cup_{\sigma<\beta} A_\sigma$ for limit ordinals β, and that there is a computable function $f(\alpha, \sigma)$ with value 1 if $\alpha \in A_\sigma$ and 0 if not.) Assume also that every σ with $A_\sigma \neq A_{\sigma+1}$ is clockable. Then for every β less than the supremum γ of the clockable ordinals, there exists a clockable limit ordinal $\tau > \beta$ such that $\varphi^{A_\tau}_{e,\tau}(x)\downarrow = 0$.*

The same would hold if we replaced "clockable" by "writable" throughout the last paragraph. However, clockability will be the property we need.

Proof. Our proof mirrors that of the Reflection Lemma for infinite-time Turing machines (Lemma 4.3 in [3]). Consider the algorithm which, on input 0, writes x and then searches and outputs the least nonclockable ordinal $\sigma > \beta$ such that $\varphi^{A_\sigma}_{e,\sigma}(x)\downarrow = 0$.

Now $A_\gamma = A$, and since $\varphi_e^A(x)\downarrow = 0$, there are plenty of ordinals $\sigma > \gamma > \beta$ for which $\varphi_{e,\sigma}^{A_\sigma}(x)\downarrow = 0$. But our algorithm runs on input 0 with no oracle, so its own halting time is clockable and greater than its output. Therefore there must exist a nonclockable stage σ between β and γ such that $\varphi_{e,\sigma}^{A_\sigma}(x)\downarrow = 0$. Let τ be the least clockable ordinal $> \sigma$. Then τ is a limit ordinal and $A_\sigma = A_\tau$, by our condition on changes to the enumeration of A, so this τ satisfies the lemma. □

Theorem 2. *There exist ORM-incomparable enumerable sets A and B of ordinals. It follows that $\emptyset <_{ORM} A <_{ORM} \emptyset^\Diamond (<_{ORM} \emptyset^\blacklozenge)$, and likewise for B.*

Proof. We build the ORM-enumerable sets A and B as follows, to satisfy the usual Friedberg-Muchnik requirements for all $e \in \omega$:

$$\mathcal{R}_e:\ (\exists x_e)[\varphi_e^A(x_e)\downarrow = 0 \text{ iff } x_e \in B] \qquad \mathcal{S}_e:\ (\exists y_e)[\varphi_e^B(y_e)\downarrow = 0 \text{ iff } y_e \in A].$$

These have the standard priority ranking: each $\mathcal{R}_e$ has higher priority than $\mathcal{S}_e$, which has higher priority than $\mathcal{R}_{e+1}$. At each stage σ we will have an approximation $x_{e,\sigma}$ to x_e, which converges to x_e as σ grows, and similarly for y_e.

Set $A_0 = B_0 = \emptyset$, and start with approximation $x_{e,0} = y_{e,0} = e$ to the witness elements x_e and y_e. We will redefine $x_{e,\sigma+1} \neq x_{e,\sigma}$ at only finitely many stages σ, namely those stages at which $\mathcal{S}$-requirements of higher priority than $\mathcal{R}_e$ act, and the same holds symmetrically for $y_{e,\sigma}$.

If τ is a limit ordinal, define $A_\tau = \cup_{\sigma<\tau} A_\sigma$, with $x_{e,\tau} = \lim_{\sigma\to\tau} x_{e,\sigma}$, and similarly for B_τ and $y_{e,\tau}$. (Notice that each $x_{e,\sigma}$ is eventually constant as σ approaches τ, so these limits exist.)

For successor ordinals $\tau = \sigma + 1$, if σ is either unclockable or a successor ordinal itself, then we preserve the settings at stage $\sigma+1$: $A_{\sigma+1} = A_\sigma$, $x_{e,\sigma+1} = x_{e,\sigma}$, and so on. Only if σ is a clockable limit ordinal do we act at the successor stage $\sigma + 1$, as follows.

At such a stage $\sigma + 1$, we fix the highest-priority requirement, say $\mathcal{R}_e$, which requires attention, by which we mean that $\mathcal{R}_e$ is not yet satisfied and $x_{e,\sigma} \leq \sigma$ and $\varphi_{e,\sigma}^{A_\sigma}(x_{e,\sigma})\downarrow = 0$. We act by enumerating $x_{e,\sigma}$ into $B_{\sigma+1}$, with $A_{\sigma+1} = A_\sigma$. We then set $x_{i,\sigma+1} = x_{i,\sigma}$ for all $i \in \omega$, and $y_{i,\sigma+1} = y_{i,\sigma}$ for all $i < e$. The lower-priority $\mathcal{S}$-requirements are injured at this stage, for we redefine

$$y_{e+j,\sigma+1} = y_{e+j,\sigma} + (x_{e,\sigma} + \sigma) + j + 1$$

for each $j \in \omega$. Notice that since $x_{e,\sigma} \leq \sigma$, we will have $B_{\sigma+1} \subseteq (\sigma + 1)$ (by induction on the preceding stages). If there is no requirement $\mathcal{R}_e$ which requires attention at stage $\sigma + 1$, then we preserve all settings from stage σ.

If the highest-priority requirement requiring attention at stage $\sigma + 1$ is an $\mathcal{S}$-requirement, say $\mathcal{S}_e$, we simply interchange A with B and the x-witnesses with the y-witnesses in the above paragraph. At this stage, we preserve $y_{i,\sigma+1} = y_{i,\sigma}$ for all i, and $x_{i,\sigma+1} = x_{i,\sigma}$ for all $i \leq e$, with

$$x_{e+j,\sigma+1} = x_{e+j,\sigma} + (y_{e,\sigma} + \sigma) + j$$

for each $j > 0$ in ω, but not for $j = 0$. This reflects the fact that $\mathcal{R}_e$ has higher priority than $\mathcal{S}_e$. Otherwise, the entire process is symmetric in A and B.

The point of the redefinition of the y-elements when we satisfy $\mathcal{R}_e$ is that since the computation $\varphi_e^{A_\sigma}(x_{e,\sigma}) = 0$ converged in $\leq \sigma$ steps, the only oracle questions it can have asked involved membership of ordinals $\leq x_{e,\sigma} + \sigma$ in the oracle set. The elements which may later enter A are the witness elements $y_{i,\rho}$ at stages $\rho > \sigma$. By redefining $y_{e+j,\sigma+1} > x_{e,\sigma} + \sigma$ for all $j \geq 0$ at stage $\sigma + 1$, we ensure that any of these elements that later enters A will not change the oracle computation $\varphi_e^A(x_{e,\sigma}) = 0$, since the computation cannot have asked whether such large elements were in A. (It is possible for $y_{e+j,\rho}$ to be redefined yet again at a stage $\rho + 1 > \sigma + 1$, but our formula also ensures that it is always redefined to be larger than it had been before.) Of course, if a higher-priority y-witness element later enters A, it could change this computation, but the usual finite-injury argument shows that eventually we will reach a stage after which no higher-priority requirement acts again. So, using induction on the requirements according to their priority, we see that for every e, the witness elements $x_e = \lim_\sigma x_{e,\sigma}$ and $y_e = \lim_\sigma y_{e,\sigma}$ exist, and that if $x_e \in B$, then $\varphi_e^A(x_e)\downarrow = 0$, and symmetrically.

A further argument is necessary for the converse, using the Reflection Lemma 3. The point of restricting our construction to clockable stages was to ensure that every witness element $x_{e,\sigma}$ and $y_{e,\sigma}$ at every stage is a writable ordinal, i.e. equal to $\varphi_p(0)$ for some program p. The clockable limit stages were chosen precisely because, being clockable, they were writable, as are their successors and the stage 0. (Also, clockability is easier to check than writability!) Then, if we redefined any $x_{i,\sigma+1}$ at stage $\sigma + 1$, we set it equal to a sum of writable ordinals, and similarly for $y_{e,\sigma+1}$. So, by induction on stages, all witness elements are writable, and thus all elements of A and B are writable. But we can write the stage at which a writable ordinal enters a semidecidable set, so $A_\gamma = A$ and $B_\gamma = B$, as required by the Reflection Lemma.

Now suppose that x_e never entered B. Then for all sufficiently large clockable limit ordinals δ, $\varphi_{e,\delta}^{A_\delta}(x_e)$ either diverges or converges to a nonzero value, so by the Reflection Lemma, the full computation $\varphi_e^A(x_e)$ cannot converge to 0. Thus again x_e witnesses that $\varphi_e^A \neq B$, so $B \not\leq_{ORM} A$. A symmetric result holds for y_e, so A and B are ORM-incomparable sets.

We have called this process a construction, and indeed it enumerated elements into A and B, without ever removing them. Nevertheless, it remains to show that A and B are actually ORM-enumerable, of course, since the description above did not use ORM's. The heart of the proof of Theorem 2 is the following lemma.

Lemma 4. *There exists an ORM which decides, for arbitrary ordinal inputs α and σ, whether $\alpha \in A_{\sigma+1} - A_\sigma$; and similarly for B. We refer to the algorithms for these machines as the* entry algorithms *for A and B.*

Proof. We use an ordinal stack machine, which is a special type of ordinal register machine. In an ordinal stack machine, along with finitely many registers, we have finitely many *stacks*, each of which (at any single stage of operation) consists of a descending (hence finite) sequence of ordinals. We can push a new ordinal from a register onto one of these stacks, provided that it is strictly less than all

ordinals currently on the stack; and we can pop the smallest ordinal off of any stack and transfer it to a register, where it can be compared to the contents of other registers, etc., by the usual register operations. In [6], Koepke and Siders use the Cantor normal form of an ordinal to prove that the functions on ordinals computable by ordinal stack machines are precisely those computable by regular ORM's, so we may prove our lemma by giving two stack-machine programs (one for A, one for B) which answer the question for arbitrary α and σ.

Our stack machine accepts the inputs α and σ. It immediately pushes σ on top of its stage stack, and pushes the ordinal $(\omega^\sigma + \alpha)$ on top of its input stack. These will stay on these stacks for the rest of the operation of this machine, but occasionally the entry algorithm will call itself and push smaller ordinals above them, which are then popped off the stack when the subroutine ends. Of course, we have access to the value σ from the top of the stage stack any time we need it, and from the two stacks together we can also compute the value of α whenever we need it.

The reason for not simply pushing α onto the input stack is that the entry algorithm may later call itself, with inputs τ and β. We will ensure that $\tau < \sigma$, but we may have to allow $\beta \geq \alpha$, in which case we could not push β onto a stack above α. However, pushing $\omega^\tau + \beta$ onto the stack above $\omega^\sigma + \alpha$ will be legal, because we will always have $\beta < \sigma$.

We give the details for the entry algorithm which decides whether α entered B at stage $\sigma + 1$. The entry algorithm for A is quite similar, of course, and in fact is used by the algorithm for B. Given α and σ, we execute the following steps.

1. If $\alpha \geq \sigma$, or if σ is not a clockable limit stage, then output 0. Otherwise, go on. (In our construction, a witness element α only enters B at successors of clockable limit stages $> \alpha$. These properties are indeed ORM-decidable.)
2. For each $e < \omega$, check whether $x_{e,\sigma} = \alpha$. If such an e exists, then go on to Step 3. Notice that if e exists, then it is unique, and we can always use it in subsequent steps by using this same process to search for it again. If no such e exists, output 0. (Only witness elements for $\mathcal{R}$-requirements ever enter B. We prove in Lemma 5 that we can compute $x_{e,\sigma}$ uniformly from e and σ.)
3. With the e from the previous step, we now simulate the operation of the program φ_e on input α with oracle A_σ for σ steps. Of course, we have no A_σ-oracle, so whenever our simulation asks whether some $\beta < \sigma$ lies in $\mathcal{A}_\sigma$, we simply use the entry algorithm to check (for all τ with $0 \leq \tau < \sigma$, starting with 0) whether β entered A at stage τ. Notice that this is allowed by our stack machine, since each such τ and β is strictly less than σ, even though the β might be $> \alpha$. (The construction ensured that $A_\sigma \subseteq \sigma$, so if the simulation asks whether some $\beta \geq \sigma$ lies in A_σ, we answer "no" immediately, without using the entry algorithm to check.) Thus we can determine whether $\varphi_{e,\sigma}^{A_\sigma}(\alpha)\downarrow = 0$. If not, output 0; if so, go on.
4. Now run the entry algorithm with α and with each stage $\tau < \sigma$, to see if $\alpha \in B_\sigma$. If so, output 0, since $\alpha \notin B_{\sigma+1} - B_\sigma$. If not, go on.
5. Now compute $x_{i,\sigma}$ for each $i < e$, and run the entry algorithm with each such $x_{i,\sigma}$ and with stage σ to determine whether any higher-priority requirement

than $\mathcal{R}_e$ enumerated any element into B at stage $\sigma + 1$. For this we do not push σ onto the stage stack, because to do so would be illegal. Nor do we pop it off that stack at the end, because we want it to stay there until the end of the run of the entry algorithm on α and σ. However, we do push $(\omega^\sigma + x_{i,\sigma})$ onto the input stack, which is legal since $x_{i,\sigma} < x_{e,\sigma}$ for $i < e$, and take it off again when we move on to the next i. If we find any such $x_{i,\sigma}$ which entered B at stage $\sigma + 1$, then output 0.

Also, we compute $y_{i,\sigma}$ for each $i < e$ and use the entry algorithm for A to determine whether any such $y_{i,\sigma}$ entered A at stage $\sigma + 1$. Here we must be careful, because this run of the entry algorithm for B might have been called by the entry algorithm for A. So we check the top (i.e. smallest) element of the stage stack from the entry algorithm for A. If that element equals σ, then we leave it there, and just push $\omega^\sigma + y_{i,\sigma}$ onto the input stack, which is safe, because if the entry algorithm for A was already running and needed to know about $x_{e,\sigma}$ entering B, it must have been asking about an element $y_{j,\sigma}$ with $j \geq e > i$, so that $\omega^\sigma + y_{j,\sigma} > \omega^\sigma + y_{i,\sigma}$.

When the entry algorithm for A concludes, we find σ on top of its stage stack and consider the top two elements, say $\beta < \delta$, on its input stack. (If there is only one element β on the input stack, then we pop β off its stack and σ off its stack, leaving nothing on either stack.) Now β must equal $\omega^\sigma + y_{i,\sigma}$. Then we pop δ off the input stack and find the largest term of its Cantor normal form, say $\delta = \omega^\tau + \theta$ for some τ and θ. If $\tau = \sigma$, then we leave δ on top of the input stack and σ on top of the stage stack. If $\tau \neq \sigma$, then we leave δ on top of the input stack, but remove σ from the stage stack. Thus, even though we left no specific indicator, when calling the entry algorithm for A, of whether we had added a new σ on top of the stage stack or not, we have still determined at the conclusion of that algorithm whether or not a new σ had been added, and if it had, then we have now removed it again.

Having completed the entry algorithm for A, we now know whether any $x_{i,\sigma}$ with $i < e$ entered A at stage $\sigma + 1$. If so, then output 0; if not, then output 1. This completes the entry algorithm for B.

(If any higher-priority requirement $\mathcal{R}_i$ or $\mathcal{S}_i$ enumerated an element into B at stage $\sigma + 1$, then $\mathcal{R}_e$ would not enumerate an element of its own. If not, then all conditions are satisfied for $\alpha = y_{e,\sigma}$ to enter B at stage $\sigma + 1$.)

We define an analogous entry algorithm to check whether an arbitrary α entered A at an arbitrary stage $\sigma + 1$, of course. Indeed, these two algorithms call each other in certain cases, as detailed in Item 5. Because $\mathcal{R}_e$ has higher priority than $\mathcal{S}_e$, the entry algorithm for A must check in Item 5 whether any $x_{i,\sigma}$ with $i \leq e$ entered B at stage $\sigma + 1$, rather than just checking for $i < e$. Otherwise, the entry algorithms are symmetric in A and B.

Both entry algorithms are computable by stack machines, and the proof that they give the correct answer is a fairly simple double induction, first on stages σ, and for each individual stage, on inputs $\alpha < \sigma$. The one twist to be noted is that we promised a separate algorithm to compute $x_{e,\sigma}$ for arbitrary e and σ;

and since this algorithm is used in the entry algorithm, it can only call the entry algorithm for smaller stages.

Lemma 5. *There are ORM's which compute $x_{e,\sigma}$ and $y_{e,\sigma}$, uniformly in e and σ, We refer to their programs as the* witness algorithms *for A and B. These ORM's use the entry algorithm from Lemma 4, but they only push stages $< \sigma$ onto the stage stack. (That is, they only ask whether elements entered A or B at stages $\tau + 1$ with $\tau < \sigma$.)*

Proof. The witness algorithm is allowed to call itself, but only with descending stages, just like the entry algorithms. To compute $y_{e,\sigma}$, we start with $e = y_{e,0}$ in the output register of a stack machine. Then go through all stages τ with $\tau+1 \leq \sigma$. At each such τ, we use the witness algorithm to compute $x_{0,\tau}, \ldots, x_{e,\tau}$ and then the entry algorithm to check whether any $\beta = x_{i,\tau}$ with $i \leq e$ entered B at stage $\tau + 1$. If so, then $y_{e,\tau+1}$ will have been redefined to equal $y_{e,\tau} + \beta + \tau + (e - i) + 1$, so we write this new value in the output register in place of $y_{e,\tau}$. Otherwise we leave the output register as it is. When $\tau + 1$ is finally $\geq \sigma$, the value in the output register will be $y_{e,\sigma}$.

Notice that the corresponding routine for computing $x_{e,\sigma}$ asks only about elements $y_{i,\tau}$ with $i < e$. This mirrors the priority ranking of the requirements, and is important for seeing that the inductive argument for correctness of this algorithm is well-founded. This proves Lemma 5, and also Lemma 4. □

With Lemma 4, it is clear that the sets A and B are ORM-enumerable. A, for instance, is the domain of the ORM-computable function which, on input α, starts with $\sigma = 0$, asks for each σ in turn whether α enters A at stage $\sigma + 1$, and halts, putting α in the domain, if it ever receives a positive answer.

ORM-enumerability immediately implies that $A \leq_{ORM} \emptyset^{\blacklozenge}$, but we promised that $A \leq_T \emptyset^{\Diamond}$. Given any ordinal x, we check first whether x is clockable. If not, then $x \notin A$. Otherwise, there is an ORM program q which writes x and then gives x as an input to the ORM-computable function whose domain is A. We can compute this q uniformly from x, and $x \in A$ iff $q \in \emptyset^{\Diamond}$. The same proof works for B, so we have a pair of ORM-incomparable semidecidable sets below $\emptyset^{\Diamond}$, as Theorem 2 claimed. □

3 Softer proofs of the result

In the preceding argument, in order to show that A and B are ORM-enumerable, we provided a detailed ORM procedure for deciding the entry algorithms for the sets A and B. We are pleased to have done so, and we believe that the procedure helps illustrate several useful techniques of ORM computation. Nevertheless, this part of the argument can be completely eliminated by making use of the main theorem of [6]. The structure of our previous argument was first to provide a set-theoretical definition of the sets A and B in terms of their approximations A_σ and B_σ, which ensured that A and B would be ORM-incomparable, and then to provide a detailed ORM algorithm to decide the entry problems $\alpha \in A_{\sigma+1} - A_\sigma$

and $\alpha \in B_{\sigma+1} - B_\sigma$, which ensured that A and B would be ORM-enumerable. The point we would like to make now is that once we have given the initial set-theoretic construction of the sets A_σ and B_σ, we can simply observe that this construction can be carried out inside Gödel's constructible universe L, in such a way that the construction is absolute to initial segments of L. That is, if some level of the constructibility hierarchy L_η satisfies that an ordinal α has entered A at stage σ, then this is truly the case. Therefore, in order to determine whether or not α enters A at stage σ, we need only search for an ordinal η such that L_η satisfies the set-theoretical assertion "α enters A at stage σ". But the question of whether L_η satisfies a given set-theoretical assertion $\varphi(\alpha, \sigma)$ is uniformly ORM-decidable from input $(\eta, \alpha, \sigma, \ulcorner\varphi\urcorner)$, by the main result of [6]. Consequently, both A and B are ORM-enumerable, as desired.

In addition, we would like to mention that an even softer argument can be made by appealing to the Sacks-Simpson [9] solution of Post's problem in α-recursion theory. It has been observed by Koepke, after the Bonn International Workshop on Ordinal Computability 2007, that for any admissible ordinal α, the subsets of α that are computable in α-recursion theory are exactly the sets that are ORM-computable in time less than α. Using the admissible ordinal γ, the supremum of the ORM-clockable ordinals, one may now transfer the solution of Post's problem from γ-recursion theory over to ordinal register machines. Koepke provides the details of this idea in [7] in the case of ordinal Turing machines, but explains how this argument also applies to ordinal register machines.

References

1. R.M. Friedberg, Two recursively enumerable sets of incomparable degrees of unsolvability, Proc. Nat. Acad. Sci. (USA) **43** (1957), 236-238.
2. J.D. Hamkins & A. Lewis, Infinite-time Turing machines, Journal of Symbolic Logic **65** 2 (2000), 567-604.
3. J.D. Hamkins & A. Lewis, Post's problem for supertasks has both positive and negative solutions, Arch. Math. Logic **41** (2002), 507-523.
4. L. Harrington & R. I. Soare, Post's Program and Incomplete Recursively Enumerable Sets, Proc. Nat. Acad. Sci. (USA) **88** (1991), 10242-10246.
5. Peter Koepke, Turing computations on ordinals, Bulletin of Symbolic Logic, **11** 3 (2005), 377–397.
6. P. Koepke & R. Bissell-Siders, Ordinal Register Machines, to appear.
7. P. Koepke, α-Recursion theory and ordinal computability, electronic manuscript.
8. A.A. Muchnik, On the Unsolvability of the Problem of Reducibility in the Theory of Algorithms, Dokl. Akad. Nauk SSSR, N.S. **109** (1956), pp. 194-197 (Russian).
9. Gerald E. Sacks and Stephen G. Simpson, The α-finite injury method, *Annals of Mathematical Logic*, 4 (1972), 343–367.
10. R.I. Soare, Recursively Enumerable Sets and Degrees (New York: Springer-Verlag, 1987).
11. P. Welch, The lengths of infinite time Turing machine computations, *Bulletin of the London Mathematical Society* **32** 2 (2000), 129-136.