LIPSCHITZ CONTINUOUS ORDINARY DIFFERENTIAL EQUATIONS ARE POLYNOMIAL-SPACE COMPLETE

AKITOSHI KAWAMURA

ABSTRACT. In answer to Ko's question raised in 1983, we show that an initial value problem given by a polynomial-time computable, Lipschitz continuous function can have a polynomial-space complete solution. The key insight is simple: the Lipschitz condition means that the feedback in the differential equation is weak. We define a class of polynomial-space computation tableaux with equally restricted feedback, and show that they are still polynomial-space complete. The same technique also settles Ko's two later questions on Volterra integral equations.

Let $g\colon [0,1] \times \mathbf{R} \to \mathbf{R}$ be a continuous function and consider the initial value problem

$$h(0) = 0, \qquad h'(t) = g\big(t, h(t)\big), \quad t \in [0,1]. \tag{1}$$

The Picard–Lindelöf (or Cauchy–Lipschitz) Theorem [15] states[1] that this equation has a unique solution $h\colon [0,1] \to \mathbf{R}$ if g is *Lipschitz continuous* (along its second argument), i. e.,

$$|g(t, y_0) - g(t, y_1)| \le Z \cdot |y_0 - y_1|, \qquad t \in [0,1],\ y_0, y_1 \in \mathbf{R} \tag{2}$$

for some constant Z independent of y_0, y_1 and t. We are interested in the computational complexity of the solution h under this condition.

Our model of computation of real functions is reviewed in Section 1. It is adopted from computable analysis and is thus consistent with the conventional notion of computability (see Section 5.3 for other perspectives on the "complexity" of similar problems). We formulate our main result in Section 2: the solution of the above equation can be polynomial-space complete, even if g is polynomial-time computable. This was open since 1983 [18]. The proof will be given in Section 3. The main idea is to regard the differential equation with the Lipschitz condition as a polynomial-space computation tableau with certain restrictions. In Section 4, we use the same techniques solve several other problems, including the ones about Volterra integral equations [20]. Section 5 discusses related results and problems.

Key words and phrases. computable analysis; computational complexity; exponential space; initial value problem; Lipschitz condition; ordinary differential equations; Picard–Lindelöf Theorem; polynomial space; Volterra integral equations.

This work was supported in part by the Nakajima Foundation and by the Natural Sciences and Engineering Research Council of Canada.

[1] There are several variants of the theorem; here is a proof sketch for ours. Let C be the set of all continuous real functions on [0, 1]. A solution of (1) is a fixed point of the operator $T\colon C \to C$ defined by

$$T(h)(t) = \int_0^t g\big(\tau, h(\tau)\big)\,\mathrm{d}\tau.$$

This solution exists and is unique by the Banach fixed point theorem, because T is a contraction with respect to the distance d given by $d(h_0, h_1) = \max_{t \in [0,1]} \exp(-2Zt)|h_0(t) - h_1(t)|$ for $h_0, h_1 \in C$.

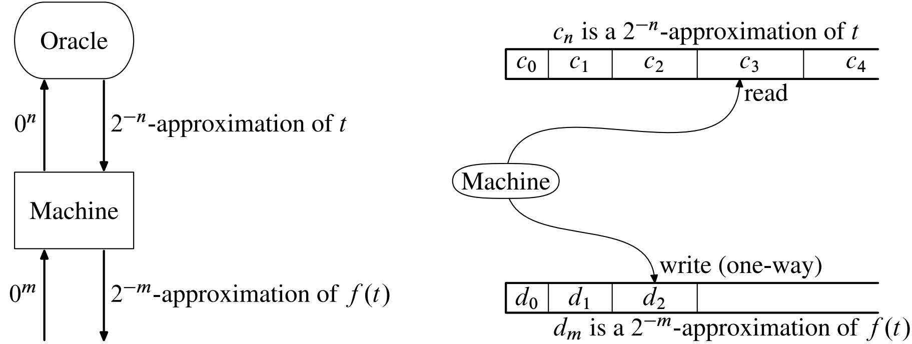

FIGURE 1. To compute a real function f, the machine should output an approximation of $f(t)$ to given precision 2^{-m} by consulting the oracle for approximations of t to any precision 2^{-n} it desires (left). An alternative picture (right) is that the machine converts any stream of improving approximations of t to a stream of improving approximations of $f(t)$.

1. COMPUTATIONAL COMPLEXITY OF REAL FUNCTIONS

The study of mathematical analysis from the viewpoint of computability is called *computable analysis* [5, 41]. We review its definitions briefly here, refining them to accommodate our complexity consideration where necessary. We assume that the reader is familiar with the basic concepts in complexity theory, such as polynomial-time and polynomial-space machines, reductions, and completeness [36]. The computability notion for real functions equivalent to ours dates back at least to Grzegorczyk [13]. Study of polynomial-time computability seems to originate in Ko and Friedman [22].

1.1. **Computing real functions.** Computers can process only finitely many bits at a time. A real number cannot be stored in its entirety, but can only be approximated. We say that a real number t is *represented* by a function A from strings to strings if for any $m \in \mathbf{N}$, the string $A(0^m)$ is the binary notation (with a sign bit at the beginning) of either $\lfloor 2^m t \rfloor$ or $\lceil 2^m t \rceil$, where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ mean rounding down and up to the nearest integer, respectively. In effect, $A(0^m)$ gives an approximation of t with precision 2^{-m} by a multiple of 2^{-m}. We also say that A is a *name* of t.

Computation of real functions is realized by *oracle Turing machines* (henceforth just *machines*) working on such representing functions A. In addition to the input, output and work tapes, the machine has a query tape and can consult an external oracle A by entering a distinguished state; the string v which is on the query tape at this moment is then replaced by $A(v)$ in one step. We write M^A for the string-to-string function computed by machine M with oracle A.

Definition 1. A machine M *computes* a function $f: [0, 1] \to \mathbf{R}$ if for any $t \in [0, 1]$ and any name A of it, M^A is a name of $f(t)$.

Thus, computation of a real function f can be thought of as a Turing reduction of (a name of) $f(t)$ to t (Fig. 1, left). A little thought shows that it can equivalently be visualized as a Turing machine that, given on the input tape an infinite sequence of approximations of t, writes approximations of $f(t)$ endlessly on the one-way output tape (Fig. 1, right).

A machine runs in *polynomial time* if there is a polynomial $p: \mathbf{N} \to \mathbf{N}$ such that, for any input string u, it halts within $p(|u|)$ steps regardless of the oracle. A function is

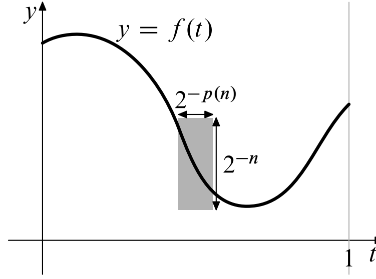

FIGURE 2. Modulus of continuity p.

(*polynomial-time*) *computable* if it is computed by some machine (that runs in polynomial time).

When writing each approximation of the output, the machine knows the input only to some finite precision. As a result, all computable functions are continuous, and all polynomial-time computable functions f have a polynomial *modulus of continuity* (Fig. 2): there is a polynomial p such that $|f(t_0) - f(t_1)| < 2^{-n}$ for all $t_0, t_1 \in [0,1]$ with $|t_0 - t_1| < 2^{-p(n)}$. In fact, it is not hard to see that polynomial-time computability can be characterized by this plus the assertion that f can be approximated at rationals:

Lemma 2. *A function $f\colon [0,1] \to \mathbf{R}$ is polynomial-time computable if and only if it has a polynomial time modulus of continuity and there is a polynomial-time computable function $g\colon ([0,1] \cap \mathbf{Q}) \times \{0\}^* \to \mathbf{Q}$ such that*

$$|g(d, 0^n) - f(d)| < 2^{-n}, \qquad d \in [0,1] \cap \mathbf{Q},\ n \in \mathbf{N}, \tag{3}$$

where rational numbers are encoded in a reasonable way.

Many familiar continuous functions are computable. For example, it is easy to see that the sine function restricted to $[0,1]$ is polynomial-time computable, because an approximation of

$$\sin t = t - \frac{t^3}{3!} + \frac{t^5}{5!} - \frac{t^7}{7!} + \cdots \tag{4}$$

to precision 2^{-m} can be found by approximating the sum of polynomially many (in m) initial terms, since this series converges fast enough on $[0,1]$.

The above definition can be straightforwardly extended to functions on compact intervals other than $[0,1]$ and on d-dimensional rectangles (by considering machines taking d oracles). The definition of polynomial-time computability can be generalized to polynomial-space, exponential-time and exponential-space computability[2], with a caveat that for space complexity, we count the query tape in[3].

[2] We say "exponential" to mean $2^{n^{O(1)}}$ (and not $2^{O(n)}$).

[3] The definition in Ko's book [19, Section 7.2.1] states to the contrary, but the subsequent theorems in the chapter in his book seem to build on the definition that does charge the query tape. On the other hand, his argument in Chapter 4 that the query tape should not be counted in discussing logarithmic space seems reasonable.

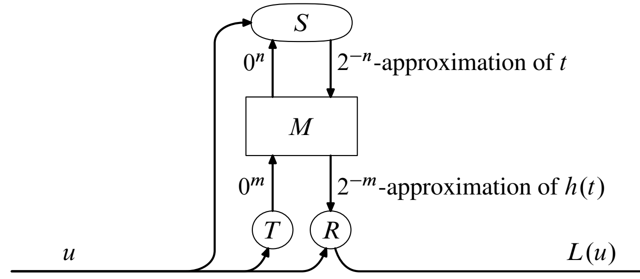

FIGURE 3. Functions R, S and T reduce L to h.

1.2. **Completeness.** We now introduce terminology to state our main results which say that certain real functions are "hard" to compute. We regard a language L as a set of strings or as a $\{0, 1\}$-valued function interchangeably, so that $L(u) = 1$ means $u \in L$. We use **P**, **NP**, **PSPACE**, **EXPTIME**, **EXPSPACE** to denote the standard classes of languages [36].

Definition 3. A function L (over strings) is said to *reduce* to an oracle machine M if there are polynomial-time computable functions R, S, T such that $L(u) = R(u, M^{S_u}(T(u)))$ for all strings u, where S_u denotes the function taking string v to $S(u, v)$. It *reduces* to a real function $h: [0, 1] \to \mathbf{R}$ if it reduces to any machine computing h (Fig. 3).

For complexity class $\mathcal{C}$, we say that a real function is *$\mathcal{C}$-hard* if all problems in $\mathcal{C}$ reduce to it. It is *polynomial-space* (resp. *exponential-space*) *complete* if it is polynomial-space (resp. exponential-space) computable and **PSPACE**-hard (resp. **EXPSPACE**-hard).

2. KO'S QUESTION

Now we return to the ordinary differential equation at the beginning of the paper. We assume the following, and ask how complex h can be:

($*$) $g: [0, 1] \times \mathbf{R} \to \mathbf{R}$ and $h: [0, 1] \to \mathbf{R}$ satisfy (1), g satisfies (2), and g is polynomial-time computable[4].

As Ko [18, Section 4] points out by analyzing the Euler method, ($*$) implies that h is polynomial-space computable (in fact, he later showed [20] that this is true for a broader class of equations than (1); see Section 4.2). From this it follows [18] that if $\mathbf{P} = \mathbf{PSPACE}$, then ($*$) implies that h is polynomial-time computable. We will prove a lower bound that matches this upper bound:

Theorem 4. *There are functions g and h satisfying ($*$) such that h is* **PSPACE***-hard.*

Corollary 5. $\mathbf{P} = \mathbf{PSPACE}$ *if ($*$) implies that h is polynomial-time computable.*

This solves the main problem in Ko's 1983 paper [18]. He had proved a partial result [18, Section 5] essentially stating that Theorem 4 holds true if the Lipschitz condition (2) in the assumption ($*$) is replaced by a weaker condition.

[4]Strictly speaking, we have defined polynomial-time computability only for functions defined on a compact rectangle. What we mean here is that the restriction of g to $[0, 1] \times [\min h, \max h]$ is polynomial-time computable. This awkward restriction could be avoided, without affecting the results in this paper, by extending Definition 1 to functions with unbounded domain, as Hoover [14] did (see also [19, pp. 57–58]). But this would slightly complicate the definition of polynomial-time computability and also make it disagree with another formulation in literature [41, Chapter 7].

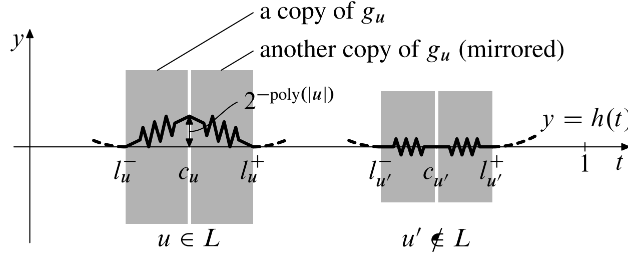

FIGURE 4. To construct g, we assign interval $[l_u^-, l_u^+]$ to each string u and put there a pair of reduced copies of g_u. The value $h(c_u)$ at the midpoint will be positive if and only if $u \in L$.

We will also consider several other variants of the question in Section 4. For more motivation, backgrounds and related problems, see Section 5.

3. PROOF OF THE THEOREM

In Section 3.1, we construct the functions g and h in Theorem 4 from a family of pairs of functions $(g_u)_u$ and $(h_u)_u$ satisfying the conditions to be specified in Lemma 6.

Section 3.2 contains the key idea of the proof. We reduce Lemma 6 to the **PSPACE**-completeness of a discrete version of the initial value problem. This discrete problem is like a **PSPACE** computation tableau, but with a certain restriction similar to the Lipschitz condition: we restrict the strength of feedback in the computation.

Section 3.3 then shows that this discrete problem is **PSPACE**-complete despite the restriction.

3.1. **Building blocks.**

To state Lemma 6, we need to extend the definition of computation in Section 1.1 to families of real functions indexed by strings u. This is done in the natural way, by giving u as another string input to the machine. For example, a family $(g_u)_u$ of functions $g_u\colon [0,1] \times [-1,1] \to \mathbf{R}$ is computed by a machine M if for any names A and B of $t \in [0,1]$ and $y \in [-1,1]$, the function that takes string 0^m to $M^{A,B}(u, 0^m)$ is a name of $g_u(t,y)$. Note that in this case, claiming that M runs in polynomial time means that it halts in time polynomial in $|u| + m$.

Lemma 6. *Let $L \in$ **PSPACE** and let $\lambda\colon \mathbf{N} \to \mathbf{N}$ be a polynomial. Then there exist a polynomial $\rho\colon \mathbf{N} \to \mathbf{N}$ and families of functions $g_u\colon [0,1] \times [-1,1] \to \mathbf{R}$ and $h_u\colon [0,1] \to [-1,1]$ indexed by binary strings u such that the family $(g_u)_u$ is polynomial-time computable and for each u we have*

(a) $h_u(t) \in [-1,1]$ *for all* $t \in [0,1]$;
(b) $g_u(0,y) = g_u(1,y) = 0$ *for all* $y \in [-1,1]$;
(c) $h_u(0) = 0$ *and* $h'_u(t) = g_u(t, h_u(t))$ *for all* $t \in [0,1]$;
(d) $|g_u(t,y_0) - g_u(t,y_1)| \le 2^{-\lambda(|u|)}|y_0 - y_1|$ *for any* $t \in [0,1]$ *and* $y_0, y_1 \in [-1,1]$;
(e) $h_u(1) = 2^{-\rho(|u|)} L(u)$.

Thus we have a family of functions g_u that each give an initial value problem whose solution h_u encodes $L(u)$ in its final value $h_u(1)$. Using this, the functions g and h in Theorem 4 can be constructed as follows: divide $[0,1)$ into infinitely many subintervals $[l_u^-, l_u^+]$, one for each u, with midpoints c_u; we put a pair of reduced copies of g_u onto $[l_u^-, c_u]$ and $[c_u, l_u^+]$ as shown in Fig. 4 so that the membership of u in L can be determined by looking at $h(c_u)$. Details are routine; see Appendix A.

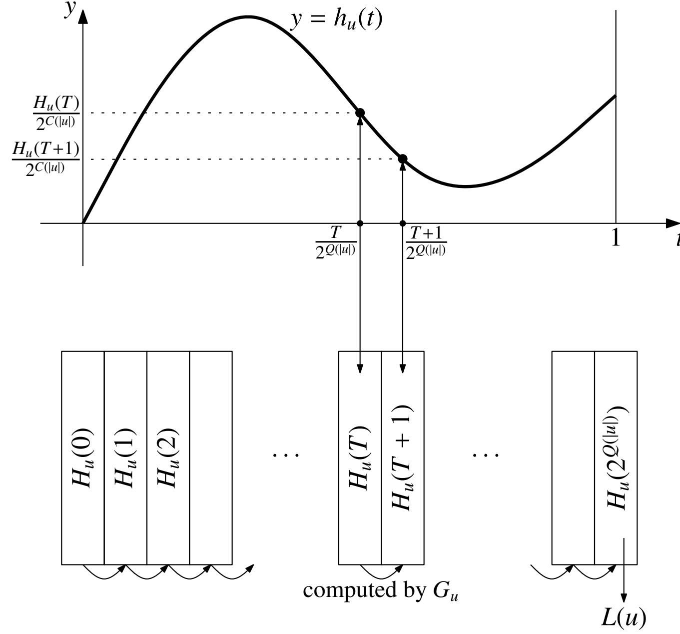

FIGURE 5. An attempt to simulate a polynomial-space Turing machine by an initial value problem is to encode the machine configuration $H_u(T)$ at each time T into the value $h_u(t) = H_u(T)/2^{C(|u|)}$ at time $t = T/2^{Q(|u|)}$.

3.2. **Discrete initial value problem and the Lipschitz condition.** An attempt to prove Lemma 6 would be as follows. Consider a polynomial-space Turing machine that decides whether a given string u belongs to L. Its configuration at each time can be encoded into a nonnegative integer less than $2^{C(|u|)}$, where C is a polynomial. There is a simple rule that maps u (the input), T (time) and d (the current configuration) to a number $G_u(T, d)$ (the next configuration) such that the recurrence

$$H_u(0) = 0, \qquad H_u(T + 1) = G_u\big(T, H_u(T)\big) \tag{5}$$

leads to $H_u(2^{Q(|u|)}) = L(u)$ for some polynomial Q. Now this situation looks similar to the one in Lemma 6: starting at 0, the value of H_u (or h_u) changes over time according to a simpler function G_u (or g_u), to reach a value eventually that indicates the answer $L(u)$. Thus we are tempted to simulate the "discrete initial value problem" (5) by embedding each value $H_u(T)$ as real number $H_u(T)/2^{C(|u|)}$ (Fig. 5).

The obstacle to this attempt is that the differential equation (c) of Lemma 6 cannot express all discrete recurrence (5): continuous trajectories cannot branch or cross one another; besides, we have the Lipschitz condition (d) that puts restriction on how strong the feedback of h to itself can be. We thus need to restrict the discrete problem (5) so that it can be simulated by the continuous version.

To do so, let us reflect on what the Lipschitz condition (d) means. A rough calculation shows that if two trajectories differ by ε at time t, then they can differ at time $t + 2^{-Q(|u|)}$ by at most $\varepsilon \exp(2^{-\lambda(|u|)} 2^{-Q(|u|)}) \approx \varepsilon(1 + 2^{-\lambda(|u|)-Q(|u|)})$. Thus, the gap can only widen (or narrow) by a factor of $\pm 2^{-\lambda(|u|)-Q(|u|)}$ at each time step of length $2^{-Q(|u|)}$. In other words, the feedback in equation (c) is so weak that each digit of h can only affect far lower digits of h at the next step.

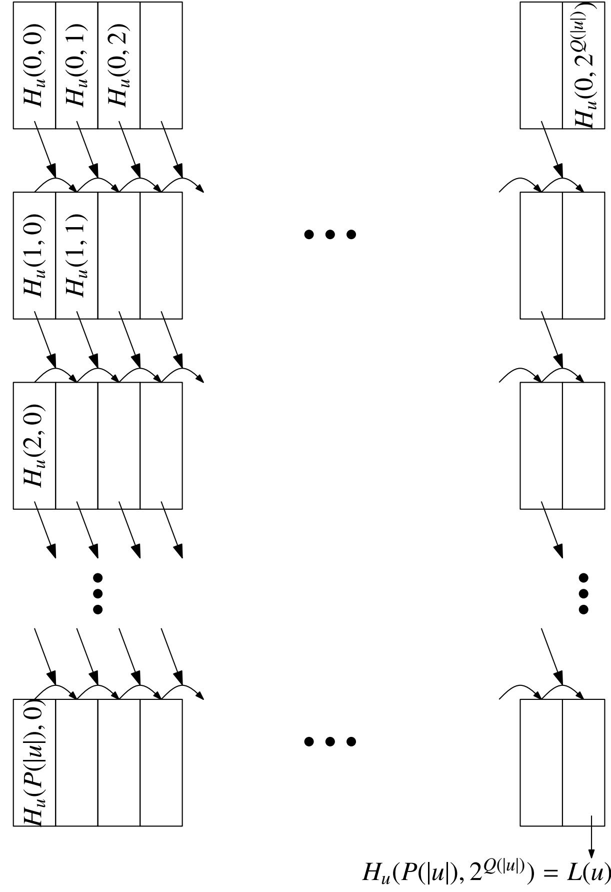

FIGURE 6. Each cell $H_u(T)$ in Fig. 5 is now divided into $H_u(0,T)$, …, $H_u(P(|u|),T)$; the increment from $H_u(i+1,T)$ to $H_u(i+1,T+1)$ is computed by G_u using the upper left cell $H_u(i,T)$.

Now we define a discrete problem that reflects this restriction. Let C, P and Q be polynomials and let

$$G_u \colon [P(|u|)] \times [2^{Q(|u|)}] \times [2^{C(|u|)}] \to \{-1,0,1\}, \tag{6}$$

$$H_u \colon [P(|u|)+1] \times [2^{Q(|u|)}+1] \to [2^{C(|u|)}], \tag{7}$$

where we write $[N] = \{0,\dots,N-1\}$ for $N \in \mathbf{N}$. Our restricted discrete initial value problem is as follows (Fig. 6):

$$H_u(i,0) = H_u(0,T) = 0, \tag{8}$$

$$H_u(i+1,T+1) = H_u(i+1,T) + G_u\big(i,T,H_u(i,T)\big). \tag{9}$$

In effect, $H_u(T)$ of (5) is now divided into components $H_u(0,T),\dots,H_u(P(|u|),T)$. We have added the restriction that G_u sees only the component $H_u(i,T)$, which in Fig. 6 means the upper left of the current cell. The next lemma states that we do not lose **PSPACE**-completeness by this restriction. Note that making G_u completely oblivious to its second argument would be an overkill, because then H_u would just add up the values of G_u, resulting in the complexity merely of #**P**.

Lemma 7. *Let $L \in$ **PSPACE**. Then there are polynomials C, P, Q and families $(G_u)_u$, $(H_u)_u$ satisfying* (6)–(9) *such that $(G_u)_u$ is polynomial-time computable and*

$$H_u(P(|u|), 2^{Q(|u|)}) = L(u) \tag{10}$$

for each string u.

Before proving this, we reduce Lemma 6 to Lemma 7 by simulating the new system (6)–(9) by the differential equation. Using G_u and H_u of Lemma 7, we construct g_u and h_u of Lemma 6 such that $h_u(T/2^{Q(|u|)}) = \sum_i H_u(i, T)/B^i$ for each T, where B is a big number. Thus, the restriction that G_u sees only the upper row corresponds to the weak feedback imposed by the Lipschitz condition.

Proof of Lemma 6. Let C, P, Q, $(G_u)_u$, $(H_u)_u$ be as in Lemma 7. By "dividing each unit time into $P(|u|)$ steps," we may assume that for each T, there is at most one i such that $G_u(i, T, Y) \neq 0$ for some Y. Write $j_u(T)$ for this unique i (define $j_u(T)$ arbitrarily if there is no such i). We may also assume that $H_u(i, 2^{Q(|u|)}) = 0$ for all $i < P(|u|)$. This can be achieved by increasing Q and extending G symmetrically with opposite sign so that it cancels out what it has done.

Let $B = 2^{C(|u|)+\lambda(|u|)+Q(|u|)+3}$. Define g_u and h_u by

$$g_u\Bigl(t, \frac{Y+\eta}{B^{j_u(T)}}\Bigr) = \frac{2^{Q(|u|)}\pi\,\sin(\theta\pi)}{2B^{j_u(T)+1}} G_u\bigl(j_u(T), T, Y \bmod 2^{C(|u|)}\bigr), \tag{11}$$

$$g_u\Bigl(t, \frac{Y-\eta}{B^{j_u(T)}}\Bigr) = (1-2\eta)\cdot g_u\Bigl(t, \frac{Y}{B^{j_u(T)}}\Bigr) + 2\eta\cdot g_u\Bigl(t, \frac{Y-1}{B^{j_u(T)}}\Bigr) \tag{12}$$

and

$$h_u(t) = \frac{1-\cos(\theta\pi)}{2}\cdot\frac{G_u\bigl(j_u(T), T, H_u(j_u(T), T)\bigr)}{B^{j_u(T)+1}} + \sum_{i=0}^{P(|u|)} \frac{H_u(i,T)}{B^i} \tag{13}$$

for all $T \in \mathbf{N}$, $\theta \in [0, 1]$, $Y \in \mathbf{Z}$, $\eta \in [0, 1/2]$ and $t = (T+\theta)/2^{Q(|u|)}$ (that make sense with (6), (7) and $g_u\colon [0,1]\times[-1,1] \to \mathbf{R}$). Note that equation (12) has g_u on both sides, but the values used in the right-hand side are already defined by (11). It is easy to verify that the definition is consistent; in particular, we use (9) to show that (13) stays the same for the two choices of T when t is a multiple of $1/2^{Q(|u|)}$.

Conditions (a) and (b) of Lemma 6 are easy. We have (e) with $\rho(k) = P(k)(C(k) + \lambda(k) + Q(k) + 3)$, since $h_u(1) = H_u(P(|u|), 2^{Q(|u|)})/B^{P(|u|)} = L(u)/B^{P(|u|)} = L(u)/2^{\rho(|u|)}$ by (13) and (10). Polynomial-time computability of $(g_u)_u$ can be verified using Lemma 2.

To see (c), observe that all terms in the right-hand side of (13) except for the summands for $i = 0, \dots, j_u(T)$ are much smaller than $1/B^{j_u(T)}$. Hence, we can write $h_u(t) = (Y+\eta)/B^{j_u(T)}$ for some $\eta \in [0, 1/2]$, where

$$Y = \sum_{i=0}^{j_u(T)} H_u(i, T)\cdot B^{j_u(T)-i}. \tag{14}$$

Since B is a multiple of $2^{C(|u|)}$, we have $Y \bmod 2^{C(|u|)} = H_u(j_u(T), T)$. Substituting these Y and η into (11), we get

$$g_u\bigl(t, h_u(t)\bigr) = \frac{2^{Q(|u|)}\pi\,\sin(\theta\pi)}{2B^{j_u(T)+1}} G_u\bigl(j_u(T), T, H_u(j_u(T), T)\bigr). \tag{15}$$

This equals $h'_u(t)$ calculated from (13).

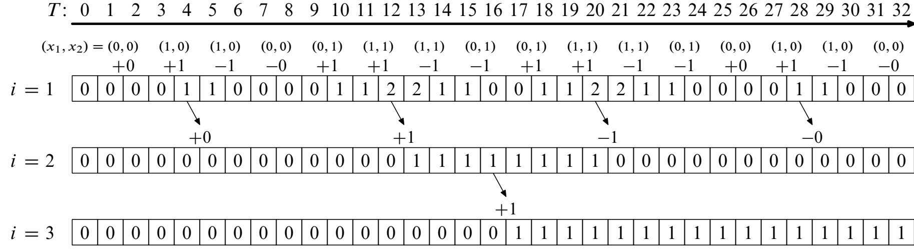

FIGURE 7. Let u be the formula $\exists x_2.\,\forall x_1.\,(x_1 \vee x_2)$, for example. The values $G_u(0, T, 0)$ encode (redundantly) the truth table of the matrix $x_1 \vee x_2$ (first branch of (18)). For example, $G_u(0, T, 0) = \pm 1$ (resp. 0) for $T = 3, 5, 27, 29$ (resp. $1, 7, 25, 31$) because $(x_1, x_2) = (1, 0)$ (resp. $(0, 0)$) makes $x_1 \vee x_2$ true (resp. false). Also observe that $H_u(1, T)$ returns to 0 every eight cells. As a result, the cell $H_u(1, 4) = 1$ (resp. $H_u(1, 12) = 2$) represents the fact that when x_2 is false (resp. true), $x_1 \vee x_2$ is satisfied by one (resp. two) of the assignments to x_1. Now look at the next row. The second branch of (18) says that for odd multiples T of 4, the values $G_u(1, T, H_u(1, T))$ are ± 1 or 0 according to whether the upper left cell has a 2 or not. Thus, they encode the smaller truth table for the subformula $\forall x_1.\,(x_1 \vee x_2)$ under each assignment to x_2. As a result, the cell $H_u(2, 16) = 1$ indicates that this subformula is satisfied by one of the assignments to x_2, which causes the last row to get incremented at $T = 17$. Observe that the final cell $H_u(3, 32)$ has a 1 precisely because the formula u is true.

For the Lipschitz condition (d), note that since the value of G_u in (11) is in $\{-1, 0, 1\}$, the difference of the two values of g_u on the right-hand side of (12) is bounded by $2 \times 2^{Q(|u|)}\pi \sin(\theta\pi)/(2B^{j_u(T)+1}) < 2^{Q(|u|)+2}/B^{j_u(T)+1}$. Thus, the slope of g_u along the second argument is at most

$$2B^{j_u(T)} \cdot \frac{2^{Q(|u|)+2}}{B^{j_u(T)+1}} = \frac{2^{Q(|u|)+3}}{B} \leq 2^{-\lambda(|u|)} \tag{16}$$

by our choice of B. □

3.3. **Discrete initial value problem is hard.** We are now left with Lemma 7. At first sight, our system (6)–(9) may look too weak to simulate a polynomial-space computation. Although we have polynomial amount of memory (rows) and exponential amount of time (columns), the "chains of dependence" of values must run from top to bottom and thus are polynomially bounded in length.

But there is a **PSPACE**-complete problem that matches well with this polynomially deep dependence: true quantified boolean formulas [36]. Evaluating a formula with n quantifiers is about evaluating a tree of depth n, with nodes labeled $\forall$ or $\exists$ at each level, assuming that the leaves (i. e., the truth table for the matrix) are easy to compute. We now simulate this tree evaluation by our system (6)–(9) (Fig. 7).

Proof of Lemma 7. We may assume that L is the set of true quantified boolean formulas. Let u be of form $Q_n x_n \ldots Q_1 x_1.\,\psi(x_1, \ldots, x_n)$, where ψ is a boolean formula and $Q_i \in \{\forall, \exists\}$ for each $i = 0, \ldots, n$. For $n - i$ bits $b_{i+1}, \ldots, b_n \in \{0, 1\}$, we write $\psi_i(b_{i+1}, \ldots, b_n) \in \{0, 1\}$ for the truth value (1 for true) of the subformula $Q_i x_i \ldots Q_1 x_1.\,\psi(x_1, \ldots, x_i, b_{i+1}, \ldots, b_n)$, so that $L(u) = \psi_n()$. We regard quantifiers as functions on

numbers (the values at numbers other than 0, 1, 2 will not matter):

$$\forall(2) = \exists(2) = \exists(1) = 1, \qquad \exists(0) = \forall(0) = \forall(1) = 0. \tag{17}$$

For $2n+1$ bits $b_0, \ldots, b_{2n} \in \{0,1\}$, we denote the number represented by them in binary by $\overline{b_{2n} \ldots b_0} = \sum_{j=0}^{2n} 2^j b_j$.

To define G_u, let

$$\begin{aligned} &G_u(i, \overline{T_{2n}T_{2n-1}\ldots T_{2i+2}T_{2i+1}100\ldots 0}, Y) \\ &\qquad = (-1)^{T_{2i+2}} \times \begin{cases} \psi_0(T_1 \oplus T_2, T_3 \oplus T_4, \ldots, T_{2n-1} \oplus T_{2n}) & \text{if } i = 0, \\ Q_i(Y) & \text{otherwise,} \end{cases} \end{aligned} \tag{18}$$

where $\oplus$ denotes the exclusive or; let $G_u(i, T, Y) = 0$ for other T (that is, when T is not an odd multiple of 2^{2i}). Define H_u from G_u by (8) and (9).

We prove by induction on $i = 0, \ldots, n$ that $H_u(i, T) \in \{0,1,2\}$ for all T (thus $C(|u|) = 2$ suffices for (7)) and that

$$G_u(i, S, H_u(i, S)) = (-1)^{S_{2i+2}} \psi_i(S_{2i+1} \oplus S_{2i+2}, \ldots, S_{2n-1} \oplus S_{2n}) \tag{19}$$

for all S of form $\overline{S_{2n}S_{2n-1}\ldots S_{2i+1}100\ldots 0}$ (it is immediate from the definition of G_u that $G_u(i, S, H_u(i, S)) = 0$ for other S). Once we have proved this, the case $i = n$ yields $G_u(n, 2^{2n}, H_u(n, 2^{2n})) = \psi_n() = L(u)$, and hence $H_u(n+1, 2^{2n}+1) = L(u)$. By adding dummy rows and columns, we have (10) for some P and Q, because $n < |u|$.

Suppose (19) as the induction hypothesis. This implies that flipping the two bits S_{2i+2} and S_{2i+1} of any $S = \overline{S_{2n}S_{2n-1}\ldots S_0}$ reverses the sign of $G_u(i, S, H_u(i, S))$. Therefore, while (8) and (9) yield

$$H_u(i+1, T) = \sum_{S=0}^{T-1} G_u\big(i, S, H_u(i, S)\big) \tag{20}$$

for each $T = 0, \ldots, 2^{Q(|u|)}$, most of the nonzero summands on the right-hand side cancel out. More precisely, if we write $T = \overline{T_{2n}T_{2n-1}\ldots T_0}$, the only terms that can survive are those that corresnpond to $S = \overline{T_{2n}T_{2n-1}\ldots T_{2i+3}00100\ldots 0}$ and to $S = \overline{T_{2n}T_{2n-1}\ldots T_{2i+3}01100\ldots 0}$. This shows that $H_u(i+1, T) \in \{0,1,2\}$.

When, in particular, $T = \overline{T_{2n}T_{2n-1}\ldots T_{2i+3}100\ldots 0}$, both of these terms survive, and hence (20) equals

$$\begin{aligned} &\psi_i(0, T_{2i+3} \oplus T_{2i+4}, \ldots, T_{2n-1} \oplus T_{2n}) \\ &\qquad\qquad + \psi_i(1, T_{2i+3} \oplus T_{2i+4}, \ldots, T_{2n-1} \oplus T_{2n}). \end{aligned} \tag{21}$$

Therefore, it follows from (17) that

$$Q_{i+1}\big(H_u(i+1, T)\big) = \psi_{i+1}(T_{2i+3} \oplus T_{2i+4}, \ldots, T_{2n-1} \oplus T_{2n}). \tag{22}$$

This and (18) yield

$$G_u(i+1, T, H_u(i+1, T)) = (-1)^{T_{2i+4}} \psi_{i+1}(T_{2i+3} \oplus T_{2i+4}, \ldots, T_{2n-1} \oplus T_{2n}), \tag{23}$$

completing the induction step. □

4. Other versions of the problem

4.1. **Complexity of the final value.** Ko discusses another version of the question which relates the complexity of the value $h(1)$, rather than the function h, to that of tally languages.

Definition 3 can be extended straightforwardly to machines taking d oracles. In particular, the case $d = 0$ means that a language L is said to reduce to a real number t if there are polynomial-time computable functions R and S such that $L(u) = R(A(S(u)))$ for any string u and any name A of t. Now we can state:

Theorem 8. *Any tally language in* **PSPACE** *reduces to* $h(1)$ *for some functions* g *and* h *satisfying* $(*)$.

Theorem 8 can be proved by arranging the building blocks from Lemma 6 in a different way. See Appendix B.

As a corollary, all tally languages of **PSPACE** are in **P** if $(*)$ implies that $h(1)$ is polynomial-time computable. This improves Ko's second main result [18, Theorem 4].

4.2. **Volterra integral equations.** Ko later studied [20] the complexity of *Volterra integral equations of the second kind*

$$h(t) = f(t) + \int_0^t g\big(t, \tau, h(\tau)\big)\,\mathrm{d}\tau, \qquad t \in [0,1], \tag{24}$$

where function h is to be solved from given functions f and g. As before, we suppose that f and g are polynomial-time computable and ask how complex h is.

If g is Lipschitz continuous (along its last argument), h is polynomial-space computable by Picard's iteration method [20, Section 3]. On the other hand, the best lower bound (in the sense of Ko's formulation, similar to Corollary 5) has been $\mathbf{P}^{\#\mathbf{P}}$, Valiant's counting class [39]. One of the two open problems in Ko [20, last paragraph of Section 1] was to close this gap. Our Theorem 4 has solved it, because the initial value problem (1) is the special case of (24) where f is constantly zero and g ignores its first argument:

Corollary 9. *There are polynomial-time computable functions* $f\colon[0,1]\to\mathbf{R}$ *and* $g\colon[0,1]\times[0,1]\times\mathbf{R}\to\mathbf{R}$ *such that* g *is Lipschitz continuous (along its last argument) and the (unique) solution of* (24) *is polynomial-space complete.*

The other problem was about the following weak version of Lipschitz continuity (2):

$$|g(t,y_0) - g(t,y_1)| \le 2^{r(n)}|y_0 - y_1|, \quad n \in \mathbf{N},\ t \in [0, 1-2^{-n}],\ y_0, y_1 \in \mathbf{R}, \tag{2$'$}$$

where r is a polynomial. Assuming $(*)$ with (2) replaced by (2′), how complex can h be, provided it has a polynomial modulus of continuity? Ko asked this question for the Volterra equation (24) (in which case the g in (2′) takes one more argument, of course), and showed that h is exponential-space computable and can be **EXPTIME**-hard. His second open problem was to close this gap.

The motivation for this problem comes from *Volterra integral equations of the first kind*, a class of equations that are considered harder to solve than (24). A common approach to solve them is to convert the equation into the form (24). This conversion does not preserve Lipschitz continuity (2) but the new equation merely satisfies (2′). See the original paper [20] for details.

We settle this problem also by a technique similar to Theorem 4. In fact, we have **EXPSPACE**-completeness even for the differential equation (1) (see Appendix C for proof):

TABLE 1. Assuming the left column plus the polynomial-time computability of g, how complex can the solution h of (1) be?

Assumptions	Positive results on h	Negative results on h
None	——	can be (non-unique and) all non-computable [1, 31, 18]
h is the unique solution	computable [28, 31, 34]	can take arbitrarily long time [25, 18]
g satisfies condition (2′)	exponential-space computable [20]	can be **EXPSPACE**-hard (our Theorem 10)
g satisfies the Lipschitz condition (2)	polynomial-space computable [8, 18]	can be **PSPACE**-hard (our Theorem 4)
g is analytic	polynomial-time computable [23, 16]	——

Theorem 10. *There are functions g and h satisfying* $(*)$ *with* (2) *replaced by* (2′) *such that h has a polynomial modulus of continuity and is* **EXPSPACE***-hard.*

5. Related results and challenges

5.1. **Other results on differential equations.** The initial value problem (1) is the most fundamental form of differential equations and has caught attention from computability and complexity theorists. Table 1 summarizes the results immediately related to ours. Other works concern the computability of different aspects of the solution [11, 17]. There is also a domain-theoretic account of this problem [9].

Computability (or not) of other classes of (partial) differential equations is also studied [32, 33, 10, 42, 7, 43, 46]. Refining these results and classifying the equations by computational complexity would be a rich source of research problems.

5.2. **Constructive versions.** A reasonable criticism about the positive results in Table 1 is that they deal with the complexity of each single solution h, whereas the practical concern for numerical analysts would be the complexity of the *operator* that "computes h from g." Is it possible to formulate such a constructive version of the positive results? If we are to keep using the oracle machine model in Section 1, this means defining a representation of real functions so that the machine can read (a name of) g and write out (a name of) h.

For computability (without time bound), this is possible: there is a canonical way to introduce a representation of functions and thus to define computability of operators [41, Chapter 6]. In fact, some of the positive results mentioned above are formulated as the computability of operators, to which the non-constructive versions are corollaries (because computable operators take computable functions to computable functions). But it is not obvious how to define complexity of operators[5]. We consider this to be an important challenge and will address it in another paper.

For negative results, however, the non-constructive formulation makes our theorems only stronger. In many situations in numerical analysis, we know the input function g not

[5] Ko proposes a definition but then argues that it is too restrictive for our purpose [19, Section 2.7].

just as a black box, but with some additional information about its behaviour. If such information can be encoded into the representation of g, it may somehow help compute h. Our non-constructive result shows that this still would not make the computation polynomial-time (unless $\mathbf{P} = \mathbf{PSPACE}$), as long as the representation assigns a name to every Lipschitz continuous functions and gives polynomial-time computable names to polynomial-time computable functions.

5.3. **Models of computation.** We have discussed complexity in the framework of computable analysis (Section 1). See the survey [21] for discussion of many other problems in this model. Although we believe that this is a natural model that characterizes what we can compute in the real world using digital computers, there are two other veins of research, totally different in philosophy, that also deal with "computability" and "complexity" of real functions.

5.3.1. *Analog models and the Church–Turing Thesis.* One is research on *analog models* [35, 26, 38, 3][6], where changes of physical quantities in the continuous world are interpreted as computation performed by an analog device or even by Nature. Not surprisingly, these models involve differential equations representing (classical) physical laws, typically (1). Relation between such analog models and the traditional notion of computation is studied by several authors. Many of them [40, 29, 42, 45] discuss possible philosophical implications on the Church–Turing Thesis: if there is a physical process whose output is computationally complex than its input, can't we then use it to build an analog computer that outperforms the Turing machine? A representative argument in defense of the Church–Turing Thesis is that, when the behaviour of the physical quantity is so wild, it cannot be reliably controlled and put into computational use. Our hierarchical results in Table 1 may suggest that there indeed is such a tradeoff between robustness and computational power: as we add stronger reasons for the system to yield a unique solution, its computational complexity drops.

5.3.2. *Algebraic models and numerical analysis.* The other approach to real computation deals with algebraic models, such as the one by Blum et al. [2]. Roughly speaking, they consider a modified version of Turing machines that store real numbers and perform operations on them in one step. The theory of information-based complexity [37, 44] uses a model closer to theirs in discussing complexity of numerical problems. The difference and connection between such algebraic models and our "bit model" are studied by several authors [4, 41, 6]. The algebraic model may be close to the intuitive picture that numerical analysts have in mind when they write pseudocodes for their algorithms, but since real numbers cannot be handled exactly in reality, reliability of the algorithm must be checked apart from the model. There is increasing interest in *validated methods* in numerical analysis that are closer to our strict model with guaranteed precision: "now that 'chips are cheap', it seems natural to shift the burden of determining the reliability of a numerical solution from the user to the computer" [27].

APPENDIX: PROOF OF THE THEOREMS

We prove Theorems 4, 8, 10 from Lemma 6. We assume that the reader has looked at the background explanation in Sections 3 and 4.

[6]See [30, 24, 12] and [17] for remediation of technical gaps in [35] and [26], respectively.

A. **Proof of Theorem 4.** Let L be a **PSPACE**-complete language. Use Lemma 6 for $\lambda(k) = 2k+2$ to obtain polynomial ρ and families $(g_u)_u$, $(h_u)_u$. Since $(g_u)_u$ is polynomial-time computable, there is a polynomial γ satisfying $|g_u(t, y)| \leq 2^{\gamma(|u|)-|u|}$. For each binary string u, let $\Lambda_u = 2^{\lambda(|u|)}$, $\Gamma_u = 2^{\gamma(|u|)}$ and

$$c_u = 1 - \frac{1}{2^{|u|}} + \frac{2\overline{u}+1}{\Lambda_u}, \qquad l_u^{\mp} = c_u \mp \frac{1}{\Lambda_u}, \tag{25}$$

where $\overline{u} \in \{0, \dots, 2^{|u|} - 1\}$ is u read as an integer in binary notation. This divides $[0, 1)$ into intervals $[l_u^-, l_u^+]$ indexed by $u \in \{0, 1\}^*$. Define

$$g\left(l_u^{\mp} \pm \frac{t}{\Lambda_u}, \frac{y}{\Lambda_u \Gamma_u}\right) = \pm \frac{g_u(t, \hat{y})}{\Gamma_u}, \tag{26}$$

$$h\left(l_u^{\mp} \pm \frac{t}{\Lambda_u}\right) = \frac{h_u(t)}{\Lambda_u \Gamma_u} \tag{27}$$

for each $t \in [0, 1]$ and $y \in \mathbf{R}$, where $\hat{y} = \max\{-1, \min\{1, y\}\}$. Let $g(1, y) = h(1) = 0$ for each $y \in \mathbf{R}$. These define g and h "seamlessly" by (b) of Lemma 6.

We show that g and h satisfy $(*)$. We begin with equation (1). It is easy to see that $h(0) = 0$ and $h'(1) = 0 = g(1, h(1))$. Since any number in $[0, 1)$ can be written in the form $l_u^{\mp} \pm t/\Lambda_u$ for some u and $t \in [0, 1]$, we have (1) by

$$\begin{aligned} h'\left(l_u^{\mp} \pm \frac{t}{\Lambda_u}\right) &= \pm \frac{h_u'(t)}{\Gamma_u} = \pm \frac{g_u(t, h_u(t))}{\Gamma_u} \\ &= g\left(l_u^{\mp} \pm \frac{t}{\Lambda_u}, \frac{h_u(t)}{\Lambda_u \Gamma_u}\right) = g\left(l_u^{\mp} \pm \frac{t}{\Lambda_u}, h\left(l_u^{\mp} \pm \frac{t}{\Lambda_u}\right)\right), \end{aligned} \tag{28}$$

where equalities are by (27), (c), (26), (27), respectively.

The Lipschitz condition (2) is satisfied with $Z = 1$ by (d) and our choice of λ. To see that g is polynomial-time computable, we use (the obvious two-dimensional generalization of) Lemma 2. When asked for a 2^{-m}-approximation of $g(T, Y)$ for rational numbers T and Y, the machine can find u, $\pm$, t, y with $(T, Y) = (l_u^{\mp} \pm t/\Lambda_u, y/\Lambda_u \Gamma_u)$ in polynomial time. Since (26) lies in $[-2^{-|u|}, 2^{-|u|}]$, the machine can safely answer 0 if $m < |u|$. Otherwise it can answer by computing $g_u(t, \hat{y})$ with precision 2^{-m}, which can be done in time polynomial in $m + |u| \leq 2m$ by the polynomial-time computability of $(g_u)_u$.

We have thus proved $(*)$. Since

$$h(c_u) = \frac{h_u(1)}{\Lambda_u \Gamma_u} = \frac{L(u)}{2^{\lambda(|u|)+\gamma(|u|)+\rho(|u|)}} \tag{29}$$

by (27) and (e), the problem L reduces to h. More precisely, the functions R, S, T in Definition 3 can be given by

$$R(u, v) = \text{the number denoted by } v, \tag{30}$$

$$S(u, 0^n) = \text{a string denoting } \lfloor 2^n c_u \rfloor, \tag{31}$$

$$T(u) = 0^{\lambda(|u|)+\gamma(|u|)+\rho(|u|)}. \tag{32}$$

Since L is **PSPACE**-complete, so is h.

B. **Proof of Theorem 8.** The idea is to arrange the building blocks g_u from Lemma 6, as we did for Theorem 4, but in a different way (Fig. 8). We again divide $[0, 1)$ into countably many intervals and put there some copies of the block g_u of Lemma 6; but this time, we

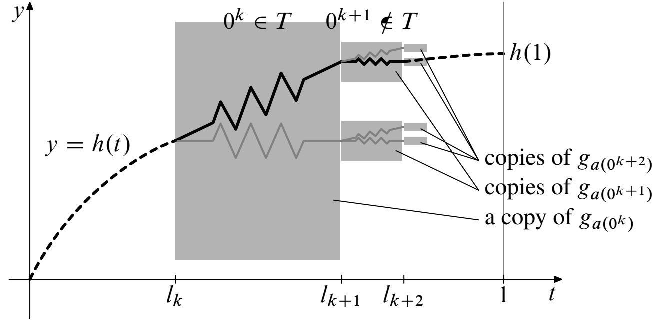

FIGURE 8. Compare with Fig. 4. The blocks $g_{a(0^k)}$ are now stacked vertically so that the tally language T can be recovered from $h(1)$.

do not put the mirror image to bring h back to 0. The values $h_u(1)$ hence pile up, so that we can recover any of them by looking at $h(1)$. We now get down to details.

Let T be a tally language in **PSPACE**. We will construct g and h satisfying $(*)$ such that T reduces to $h(1)$. Let L be a **PSPACE**-complete language. There is a polynomial-time computable string function a with $T(0^k) = L(a(0^k))$ for all $k \in \mathbf{N}$. Apply Lemma 6 to $\lambda(k) = k + 1$ to obtain the polynomial ρ and the families $(g_u)_u$, $(h_u)_u$. Since $(g_u)_u$ is polynomial-time computable, there is a monotone polynomial $\gamma\colon \mathbf{N} \to \mathbf{N}$ satisfying $|g_{a(0^k)}(t, y)| \le 2^{\gamma(k)}$ for each k. Let $l_k = 1 - 2^{-k}$ and define

$$g\left(l_k + \frac{t}{2^{k+1}}, \frac{2j + (-1)^j y}{2^{2k+\gamma(k)+\overline{\rho}(k)}}\right) = \frac{g_{a(0^k)}(t, y)}{2^{k-1+\gamma(k)+\overline{\rho}(k)}}, \tag{33}$$

$$h\left(l_k + \frac{t}{2^{k+1}}\right) = \frac{h_{a(0^k)}(t)}{2^{2k+\gamma(k)+\overline{\rho}(k)}} + \sum_{\kappa=0}^{k-1} \frac{T(0^\kappa)}{2^{2\kappa+\gamma(\kappa)+\overline{\rho}(\kappa)+\rho(\kappa)}} \tag{34}$$

for each $k \in \mathbf{N}$, $t \in [0, 1]$, $y \in [-1, 1]$ and $j \in \mathbf{Z}$, where $\overline{\rho}(k) = \rho(0) + \cdots + \rho(k-1)$. Complete the definition by $g(1, y) = 0$ and

$$h(1) = \sum_{k=0}^{\infty} \frac{T(0^k)}{2^{2k+\gamma(k)+\overline{\rho}(k)+\rho(k)}}. \tag{35}$$

By (35), the language T reduces to $h(1)$.

We show that these g and h satisfy $(*)$. Well-definedness and Lipschitz continuity of g follow from (b) and (d), similarly to the proof of Theorem 4. Polynomial-time computability also follows from that of $(g_u)_u$ again. Since all terms under the summation symbol in (34) are divisible by $4/2^{2k+\gamma(k)+\overline{\rho}(k)}$, substituting (34) into the second argument of (33) yields

$$\begin{aligned} g\left(l_k + \frac{t}{2^{k+1}}, h\left(l_k + \frac{t}{2^{k+1}}\right)\right) &= \frac{g_{a(0^k)}\big(t, h_{a(0^k)}(t)\big)}{2^{k-1+\gamma(k)+\overline{\rho}(k)}} \\ &= \frac{h'_{a(0^k)}(t)}{2^{k-1+\gamma(k)+\overline{\rho}(k)}} = h'\left(l_k + \frac{t}{2^{k+1}}\right), \end{aligned} \tag{36}$$

where the second and third equalities are by (c) and (34), respectively. This and $h'(1) = 0 = g(h(1))$ yield (1). We have thus proved $(*)$.

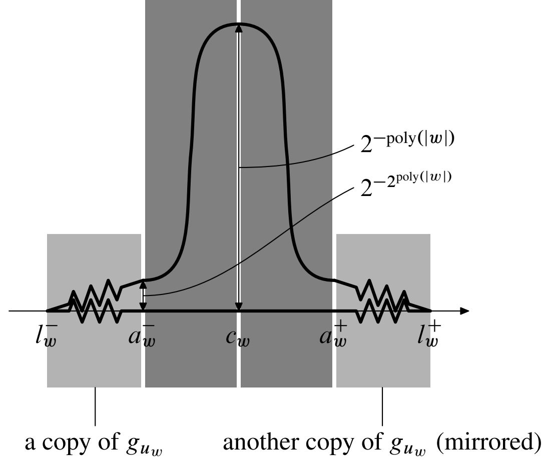

FIGURE 9. The construction for Theorem 10 is similar to Fig. 4, but this time we put an "amplifier" in the middle. Then $w \in L$ if and only if $h(c_w)$ is positive, in which case it is only polynomially small in $|w|$.

C. **Proof of Theorem 10.** Let $L \in$ **PSPACE** be the set of triples $(M, x, 0^s)$ such that M encodes a Turing machine that, on input x, uses space no more than s and accepts. For each triple $w = (M, x, s)$, let $u_w = (M, x, 0^s)$. Under a suitable encoding of triples into strings, $|u_w| \le 2^{|w|}$. It is easy to see that $L' = \{\, w : u_w \in L \,\}$ is **EXPSPACE**-complete.

The idea is that, as we did for Theorem 4, we will divide $[0, 1)$ into infinitely many intervals $[l_w^-, l_w^+]$ with midpoints c_w, and put there the functions g_{u_w} of Lemma 6 to compute whether $w \in L'$. But this time, the "output" $h_{u_w}(1)$ is exponentially small in $|w|$, so we will need "amplifiers" to make the value $h(c_w)$ visibly large (Fig. 9). Because we use stronger and stronger amplifiers as $|w| \to \infty$, the function g will not satisfy the full Lipschitz condition (2), but it still satisfies the relaxed condition (2′).

Use Lemma 6 for $\lambda(k) = 0$ to obtain polynomial ρ and families $(g_u)_u$, $(h_u)_u$. Since $(g_u)_u$ is polynomial-time computable, there is a polynomial γ satisfying $|g_u(t, y)| \le 2^{\gamma(|u|)-|u|}$. We may assume that $\gamma(k)-k$ is strictly increasing in k and that $(1.5 \ln 2)\gamma(k) \le 2^{\gamma(k)-k}$ for all k. For each binary string w, let $\Lambda_w = 2^{2|w|+3}$, $\Gamma_w = 2^{\gamma(2^{|w|})}$ and

$$c_w = 1 - \frac{1}{2^{|w|}} + \frac{4\overline{w} + 2}{\Lambda_w}, \qquad a_w^{\mp} = c_w \mp \frac{1}{\Lambda_w}, \qquad l_w^{\mp} = c_w \mp \frac{2}{\Lambda_w}, \tag{37}$$

where $\overline{w} \in \{0, \dots, 2^{|w|} - 1\}$ is w read as an integer in binary notation. This divides $[0, 1)$ into intervals $[l_w^-, l_w^+]$ of length $4/\Lambda_w$ indexed by $w \in \{0, 1\}^*$. Define

$$g\left(l_w^{\mp} \pm \frac{t}{\Lambda_w}, \frac{y}{\Lambda_w \Gamma_w}\right) = \pm \frac{g_{u_w}(t, \hat{y})}{\Gamma_w}, \tag{38}$$

$$g\left(a_w^{\mp} \pm \frac{t}{\Lambda_w}, Y\right) = \pm 6t(1-t)Y\Lambda_w \ln \Gamma_w, \tag{39}$$

$$h\left(l_w^{\mp} \pm \frac{t}{\Lambda_w}\right) = \frac{h_{u_w}(t)}{\Lambda_w \Gamma_w}, \tag{40}$$

$$h\left(a_w^{\mp} \pm \frac{t}{\Lambda_w}\right) = \frac{\Gamma_w^{t^2(3-2t)} h_{u_w}(1)}{\Lambda_w \Gamma_w} \tag{41}$$

for each $t \in [0,1]$ and $y, Y \in \mathbf{R}$, where $\hat{y} = \max\{-1, \min\{1, y\}\}$. Let $g(1, Y) = h(1) = 0$ for each $Y \in \mathbf{R}$. This defines g and h seamlessly by (b) of Lemma 6. We show that g and h satisfy $(*)$ with (2) replaced by (2′).

For the equation (1), it is again easy to see that $h(0) = 0$ and $h'(1) = 0 = g(1, h(1))$. Numbers in $[0, 1)$ can be written either as $l_w^{\mp} \pm t/\Lambda_w$ or as $a_w^{\mp} \pm t/\Lambda_w$, and for them the equation follows by

$$h'\left(l_w^{\mp} \pm \frac{t}{\Lambda_w}\right) = \pm\frac{h'_{u_w}(t)}{\Gamma_w} = \pm\frac{g_{u_w}(t, h_{u_w}(t))}{\Gamma_w} \tag{42}$$

$$= g\left(l_w^{\mp} \pm \frac{t}{\Lambda_w}, \frac{h_{u_w}(t)}{\Lambda_w \Gamma_w}\right) = g\left(l_w^{\mp} \pm \frac{t}{\Lambda_w}, h\left(l_w^{\mp} \pm \frac{t}{\Lambda_w}\right)\right),$$

$$h'\left(a_w^{\mp} \pm \frac{t}{\Lambda_w}\right) = \pm\frac{6t(1-t)\ln \Gamma_w}{\Gamma_w} \Gamma_w^{t^2(3-2t)} h_{u_w}(1) \tag{43}$$

$$= g\left(a_w^{\mp} \pm \frac{t}{\Lambda_w}, \frac{\Gamma_w^{t^2(3-2t)} h_{u_w}(1)}{\Lambda_w \Gamma_w}\right) = g\left(a_w^{\mp} \pm \frac{t}{\Lambda_w}, h\left(a_w^{\mp} \pm \frac{t}{\Lambda_w}\right)\right),$$

where we used (40), (c), (38), (40), (41), (39), (41) for each equality.

The condition (2′) is satisfied with $r(k) = 2k + 3 + s(k)$, where s is any polynomial such that $2^{s(k)} \geq (1.5 \ln 2)\gamma(2^k)$. For if $T \in [l_w^-, a_w^-]$ or $T \in [a_w^+, l_w^+]$ for some w, then by (d) and (38), we have

$$\frac{|g(T, Y_0) - g(T, Y_1)|}{|Y_0 - Y_1|} \leq \frac{2^{-\lambda(|u_w|)}}{\Gamma_w} \Lambda_w \Gamma_w \leq \Lambda_w = 2^{2|w|+3} \leq 2^{r(|w|)}. \tag{44}$$

If $T \in [a_w^-, a_w^+]$ for some w, then by (39) we have

$$\frac{|g(T, Y_0) - g(T, Y_1)|}{|Y_0 - Y_1|} \leq 2^{2|w|+3} \cdot 1.5 \ln \Gamma_w \tag{45}$$

$$= 2^{2|w|+3} \cdot 1.5\gamma(2^{|w|}) \ln 2 \leq 2^{2|w|+3} \cdot 2^{s(|w|)} = 2^{r(|w|)}.$$

To see that g is polynomial-time computable, we use the characterization in (the obvious two-dimensional generalization of) Lemma 2. Suppose we are asked for an approximation of $g(T, Y)$ to precision 2^{-m} for some $T \in [0, 1] \cap \mathbf{Q}$ and $Y \in [-1, 1] \cap \mathbf{Q}$. We first find a string w and $t \in [0, 1] \cap \mathbf{Q}$ such that T can be written as $l_w^{\mp} \pm t/\Lambda_w$ or as $a_w^{\mp} \pm t/\Lambda_w$. In the latter case, it is easy to compute the desired approximation using (39). In the former case, we use (38) as follows:

- If $m < 2^{|w|}$, we can safely answer 0, because the value (38) is in $[-2^{-2^{|w|}}, 2^{-2^{|w|}}]$ by $|g_{u_w}(t, \hat{y})| \leq 2^{\gamma(|u_w|) - |u_w|} \leq 2^{\gamma(2^{|w|}) - 2^{|w|}}$.
- Otherwise, we compute $\hat{y} \in \mathbf{Q}$, where $y = \Lambda_w \Gamma_w Y$, and then get the desired approximation of (38) by computing $g_{u_w}(t, \hat{y})$ to an appropriate precision. This can be done, by the polynomial-time computability of $(g_u)_u$, in time polynomial in m and $|u_w|$. But this is in fact polynomial in m, since $|u_w| \leq 2^{|w|} \leq m$.

We have thus proved $(*)$ with (2) replaced by (2′). Since (41) yields

$$h(c_w) = \frac{h_{u_w}(1)}{\Lambda_w} = \frac{L(u_w)}{2^{2|w|+3+\rho(|w|)}}, \tag{46}$$

the language L' is reduced to h. Since L' is **EXPSPACE**-complete, h is **EXPSPACE**-hard.

Finally, we claim that h has a polynomial modulus of continuity. Precisely, we will show that $|h(T_0) - h(T_1)| < 2^{-k}$ whenever $k \in \mathbf{N}$ and $0 \leq T_1 - T_0 < 2^{-s(k)+k}$. We may assume

that T_0 and T_1 are both greater than $1 - 2^{-k}$ or both smaller. In the first case, we have $|h(T_0) - h(T_1)| \le |h(T_0)| + |h(T_1)| \le 1/2^{2k+3} + 1/2^{2k+3} < 2^{-k}$ by (40), (41) and (a). In the second case, each point $T \in [T_0, T_1]$ belongs to $[l_w^-, l_w^+]$ for some string w of length less than k. If $T \in [l_w^-, a_w^-] \cup [a_w^+, l_w^+]$, then $|h'(T)| \le 2^{\gamma(|u_w|)}/\Gamma_w \le 1$ by the first line of (42); otherwise, $|h'(T)| \le 6(1/4)(\ln \Gamma_w) h_{u_w}(1) = (1.5 \ln 2)\gamma(2^{|w|}) h_{u_w}(1) \le 2^{s(|w|)} \le 2^{s(k)}$ by the first line of (43). We thus have $|h'(T)| \le 2^{s(k)}$, and hence $|h(T_0) - h(T_1)| \le 2^{s(k)}(T_1 - T_0) < 2^{s(k)} \cdot 2^{-s(k)+k} = 2^{-k}$.

Acknowledgement

The author is grateful to Stephen A. Cook, Ker-I Ko and the anonymous referees for comments on the manuscript that helped improve the presentation.

References

[1] O. Aberth, "The failure in computable analysis of a classical existence theorem for differential equations," *Proceedings of the American Mathematical Society*, vol. 30, pp. 151–156, 1971.

[2] L. Blum, F. Cucker, M. Shub, and S. Smale, *Complexity and Real Computation*. Springer, 1997.

[3] O. Bournez and M. L. Campagnolo, "A survey on continuous time computations," in *New Computational Paradigms*. Springer, 2008, pp. 383–423.

[4] V. Brattka and P. Hertling, "Feasible real random access machines," *Journal of Complexity*, vol. 14, pp. 490–526, 1998.

[5] V. Brattka, P. Hertling, and K. Weihrauch, "A tutorial on computable analysis," in *New Computational Paradigms: Changing Conceptions of What is Computable*, S. B. Cooper, B. Löwe, and A. Sorbi, Eds. Springer, 2008, pp. 425–491.

[6] M. Braverman, "On the complexity of real functions," in *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, 2005, pp. 155–164.

[7] D. Cenzer and J. B. Remmel, "Index sets for computable differential equations," *Mathematical Logic Quarterly*, vol. 50, no. 4-5, pp. 329–344, 2004.

[8] J. Cleave, "The primitive recursive analysis of ordinary differential equations and the complexity of their solutions," *Journal of Computer and System Sciences*, vol. 3, pp. 447–455, 1969.

[9] A. Edalat and D. Pattinson, "A domain theoretic account of Picard's theorem," in *Proceedings of the 31st International Colloquium on Automata, Languages and Programming*, ser. Lecture Notes in Computer Science, vol. 3142, 2004, pp. 494–505.

[10] W. Gay, B. Zhang, and N. Zhong, "Computability of solutions of the Korteweg–de Vries equation," *Mathematical Logic Quarterly*, vol. 47, no. 1, pp. 93–110, 2001.

[11] D. S. Graça, J. Buescu, and M. L. Campagnolo, "Boundedness of the domain of definition is undecidable for polynomial ODEs," in *Proceedings of the Fourth International Conference of Computability and Complexity in Analysis*, 2007, pp. 127–135.

[12] D. S. Graça, "Some recent developments on Shannon's General Purpose Analog Computer," *Mathematical Logic Quarterly*, vol. 50, pp. 473–485, 2004.

[13] A. Grzegorczyk, "Computable functionals," *Fundamenta Mathematicae*, vol. 42, pp. 168–202, 1955.

[14] H. J. Hoover, "Feasible real functions and arithmetic circuits," *SIAM Journal on Computing*, vol. 19, no. 1, pp. 182–204, 1990.

[15] P. Hsieh and Y. Sibuya, *Basic Theory of Ordinary Differential Equations*. Springer, 1999.

[16] A. Kawamura, Manuscript.

[17] ——, "Differential recursion," *ACM Transactions on Computational Logic*, vol. 10, no. 3, article 22, 2009.

[18] K. Ko, "On the computational complexity of ordinary differential equations," *Information and Control*, vol. 58, pp. 157–194, 1983.

[19] ——, *Computational Complexity of Real Functions*. Birkhäuser Boston, 1991.

[20] ——, "On the computational complexity of integral equations," *Annals of Pure and Applied Logic*, vol. 58, no. 3, pp. 201–228, 1992.

[21] ——, "Polynomial-time computability in analysis," in *Handbook of Recursive Mathematics: Volume 2: Recursive Algebra, Analysis and Combinatorics*, ser. Studies in Logic and the Foundations of Mathematics, I. L. Ershov *et al.*, Eds. North-Holland, 1998, vol. 139, pp. 1271–1317.

[22] K. Ko and H. Friedman, "Computational complexity of real functions," *Theoretical Computer Science*, vol. 20, no. 3, pp. 323–352, 1982.
[23] ——, "Computing power series in polynomial time," *Advances in Applied Mathematics*, vol. 9, pp. 40–50, 1988.
[24] L. Lipshitz and L. A. Rubel, "A differentially algebraic replacement theorem, and analog computability," *Proceedings of the American Mathematical Society*, vol. 99, no. 2, pp. 367–372, 1987.
[25] W. Miller, "Recursive function theory and numerical analysis," *Journal of Computer and System Sciences*, vol. 4, pp. 465–472, 1970.
[26] C. Moore, "Recursion theory on the reals and continuous-time computation," *Theoretical Computer Science*, vol. 162, pp. 23–44, 1996.
[27] N. S. Nedialkov, K. R. Jackson, and G. F. Corliss, "Validated solutions of initial value problems for ordinary differential equations," *Applied Mathematics and Computation*, vol. 105, pp. 21–68, 1999.
[28] W. F. Osgood, "Beweis der Existenz einer Lösung der Differentialgleichung $\frac{dy}{dx} = f(x, y)$ ohne Hinzunahme der Cauchy-Lipschitz'schen Bedingung," *Monatshefte für Mathematik*, vol. 9, no. 1, pp. 331–345, 1898.
[29] R. Penrose, *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics*. New York, NY, USA: Oxford University Press, Inc., 1989.
[30] M. B. Pour-El, "Abstract computability and its relation to the General Purpose Analog Computer (some connections between logic, differential equations and analog computers)," *Transactions of the American Mathematical Society*, vol. 199, pp. 1–28, 1974.
[31] M. B. Pour-El and I. Richards, "A computable ordinary differential equation which possesses no computable solution," *Annals of Mathematical Logic*, vol. 17, no. 1–2, pp. 61–90, 1979.
[32] ——, "The wave equation with computable initial data such that its unique solution is not computable," *Advances in Mathematics*, vol. 39, no. 3, pp. 215–239, 1981.
[33] M. B. Pour-El and N. Zhong, "The wave equation with computable initial data whose unique solution is nowhere computable," *Mathematical Logic Quarterly*, vol. 43, pp. 499–509, 1997.
[34] K. Ruohonen, "An effective Cauchy–Peano existence theorem for unique solutions," *International Journal of Foundations of Computer Science*, vol. 7, no. 2, pp. 151–160, 1996.
[35] C. E. Shannon, "Mathematical theory of the Differential Analyzer," *Journal of Mathematics and Physics*, vol. 20, no. 4, pp. 337–354, 1941.
[36] M. Sipser, *Introduction to the Theory of Computation*, 2nd ed. Course Technology, 2005.
[37] J. F. Traub, G. W. Wasilkowski, and H. Woźniakowski, *Information-Based Complexity*. Academic Press, 1988.
[38] J. V. Tucker and J. I. Zucker, "Computability of analog networks," *Theoretical Computer Science*, vol. 371, no. 1-2, pp. 115–146, 2007.
[39] L. G. Valiant, "The complexity of computing the permanent," *Theoretical Computer Science*, vol. 8, pp. 189–201, 1979.
[40] A. Vergis, K. Steiglitz, and B. Dickinson, "The complexity of analog computation," *Mathematics and Computers in Simulation*, vol. 28, no. 2, pp. 91–113, 1986.
[41] K. Weihrauch, *Computable Analysis: An Introduction*, ser. Texts in Theoretical Computer Science. Springer, 2000.
[42] K. Weihrauch and N. Zhong, "Is wave propagation computable or can wave computers beat the Turing machine?" *Proceedings of the London Mathematical Society*, vol. 85, no. 2, pp. 312–332, 2002.
[43] ——, "Computing Schrödinger propagators on type-2 Turing machines," *Journal of Complexity*, vol. 22, no. 6, pp. 918–935, 2006.
[44] A. G. Werschulz, *The Computational Complexity of Differential and Integral Equations: An Information-Based Approach*. Oxford University Press, 1991.
[45] A. C. Yao, "Classical physics and the Church–Turing Thesis," *Journal of the Association for Computing Machinery*, vol. 50, no. 1, pp. 100–105, 2003.
[46] N. Zhong, "Computable analysis of a boundary-value problem for the Korteweg–de Vries equation," *Theory of Computing Systems*, vol. 41, no. 1, pp. 155–175, 2007.

DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF TORONTO, 10 KING'S COLLEGE ROAD, TORONTO, ON, CANADA M5S 3G4
E-mail address: kawamura@cs.toronto.edu