Computational Complexity of Smooth Differential Equations

Akitoshi Kawamura[1], Hiroyuki Ota[1], Carsten Rösnick[2], and Martin Ziegler[2]

[1] University of Tokyo
[2] TU Darmstadt

Abstract. The computational complexity of the solution h to the ordinary differential equation $h(0) = 0$, $h'(t) = g(t, h(t))$ under various assumptions on the function g has been investigated in hope of understanding the intrinsic hardness of solving the equation numerically. Kawamura showed in 2010 that the solution h can be PSPACE-hard even if g is assumed to be Lipschitz continuous and polynomial-time computable. We place further requirements on the smoothness of g and obtain the following results: the solution h can still be PSPACE-hard if g is assumed to be of class C^1; for each $k \geq 2$, the solution h can be hard for the counting hierarchy if g is of class C^k.

1 Introduction

Let $g\colon [0,1] \times \mathbf{R} \to \mathbf{R}$ be continuous and consider the differential equation

$$h(0) = 0, \qquad Dh(t) = g(t, h(t)) \quad t \in [0,1], \tag{1}$$

where Dh denotes the derivative of h. How complex can the solution h be, assuming that g is polynomial-time computable? Here, polynomial-time computability and other notions of complexity are from the field of *Computable Analysis* [16] and measure how hard it is to approximate real functions with specified precision (Section 2).

If we put no assumption on g other than being polynomial-time computable, the solution h (which is not unique in general) can be non-computable. Table 1 summarizes known results about the complexity of h under various assumptions (that get stronger as we go down the table). In particular, if g is (globally) Lipschitz continuous, then the (unique) solution h is known to be polynomial-space computable but still can be PSPACE-hard [4]. In this paper, we study the complexity of h when we put stronger assumptions about the smoothness of g.

In numerical analysis, knowledge about smoothness of the input function (such as being differentiable enough times) is often beneficial in applying certain algorithms or simplifying their analysis. However, to our knowledge, this casual understanding that smoothness is good has not been rigorously substantiated in terms of computational complexity theory. This motivates us to ask whether, for our differential equation (1), smoothness really reduces the complexity of the solution.

Table 1. The complexity of the solution h of (1) assuming g is polynomial-time computable.

Assumptions	Upper bounds	Lower bounds
—	—	can be all non-computable [13]
h is the unique solution	computable [2]	can take arbitrarily long time [8, 11]
the Lipschitz condition	polynomial-space [8]	can be PSPACE-hard [4]
g is of class $\mathrm{C}^{(\infty,1)}$	polynomial-space	can be PSPACE-hard (Theorem 1)
g is of class $\mathrm{C}^{(\infty,k)}$ (for each constant k)	polynomial-space	can be CH-hard (Theorem 2)
g is analytic	polynomial-time [3, 10, 12]	—

One extreme is the case where g is analytic: h is then polynomial-time computable (the last row of the table) by an argument based on Taylor series[3] (this does not necessarily mean that computing the values of h from those of g is easy; see the last paragraph of Section 4). Thus our interest is in the cases between Lipschitz and analytic (the fourth and fifth rows). We say that g is of class $\mathrm{C}^{(i,j)}$ if the partial derivative $D^{(n,m)}g$ (often also denoted $\partial^{n+m}g(t,y)/\partial t^n \partial y^m$) exists and is continuous for all $n \le i$ and $m \le j$;[4] it is said to be of class $\mathrm{C}^{(\infty,j)}$ if it is of class $\mathrm{C}^{(i,j)}$ for all $i \in \mathbf{N}$.

Theorem 1. *There is a polynomial-time computable function* $g\colon [0,1]\times[-1,1] \to \mathbf{R}$ *of class* $\mathrm{C}^{(\infty,1)}$ *such that the equation* (1) *has a* PSPACE*-hard solution* $h\colon [0,1] \to \mathbf{R}$.

Theorem 2. *Let k be a positive integer. There is a polynomial-time computable function* $g\colon [0,1] \times [-1,1] \to \mathbf{R}$ *of class* $\mathrm{C}^{(\infty,k)}$ *such that the equation* (1) *has a* CH*-hard solution* $h\colon [0,1] \to \mathbf{R}$*, where* CH $\subseteq$ PSPACE *is the Counting Hierarchy (see Section 3.2).*

We said $g\colon [0,1]\times[-1,1] \to \mathbf{R}$ instead of $g\colon [0,1]\times\mathbf{R} \to \mathbf{R}$, because the notion of polynomial-time computability of real functions in this paper is defined only when the domain is a bounded closed region.[5] This makes the equation (1) ill-defined in case h ever takes a value outside $[-1,1]$. By saying that h is a solution

[3] As shown by Müller [12] and Ko and Friedman [10], polynomial-time computability of an analytic function on a compact interval is equivalent to that of its Taylor sequence at a point (although the latter is a local property, polynomial-time computability on the whole interval is implied by analytic continuation; see [12, Corollary 4.5] or [3, Theorem 11]). This implies the polynomial-time computability of h, since we can efficiently compute the Taylor sequence of h from that of g.

[4] Another common terminology is to say that g is of class C^k if it is of class $\mathrm{C}^{(i,j)}$ for all i, j with $i + j \le k$.

[5] Although we could extend our definition to functions with unbounded domain [6, Section 4.1], the results in Table 1 do not hold as they are, because polynomial-

in Theorem 1, we are also claiming that $h(t) \in [-1, 1]$ for all $t \in [0, 1]$. In any case, since we are putting stronger assumptions on g than Lipschitz continuity, such a solution h, if it exists, is unique.

Whether smoothness of the input function reduces the complexity of the output has been studied for operators other than solving differential equations, and the following negative results are known. The integral of a polynomial-time computable real function can be #P-hard, and this does not change by restricting the input to C^∞ (infinitely differentiable) functions [9, Theorem 5.33]. Similarly, the function obtained by maximization from a polynomial-time computable real function can be NP-hard, and this is still so even if the input function is restricted to C^∞ [9, Theorem 3.7][6]. (Restricting to analytic inputs renders the output polynomial-time computable, again by the argument based on Taylor series.) In contrast, for the differential equation we only have Theorem 2 for each k, and do not have any hardness result when g is assumed to be infinitely differentiable.

Theorems 1 and 2 are about the complexity of each solution h. We can also talk about the complexity of the operator that maps g to h; see Section 4.

Notation Let $\mathbf{N}$, $\mathbf{Z}$, $\mathbf{Q}$, $\mathbf{R}$ denote the set of natural numbers, integers, rational numbers and real numbers, respectively.

Let A and B be bounded closed intervals in $\mathbf{R}$. We write $|f| = \sup_{x \in A} f(x)$ for $f \colon A \to \mathbf{R}$. A function $f \colon A \to \mathbf{R}$ is *of class* C^i (i-times continuously differentiable) if all the derivatives $Df, D^2f, \dots, D^if$ exist and are continuous.

For a differentiable function g of two variables, we write $D_1 g$ and $D_2 g$ for the derivatives of g with respect to the first and the second variable, respectively. A function $g \colon A \times B \to \mathbf{R}$ is of *class* $C^{(i,j)}$ if for each $n \in \{0, \dots, i\}$ and $m \in \{0, \dots j\}$, the derivative $D_1^n D_2^m g$ exists and is continuous. A function g is of *class* $C^{(\infty,j)}$ if it is of class $C^{(i,j)}$ for all $i \in \mathbf{N}$. When g is of class $C^{(i,j)}$, we write $D^{(i,j)}g$ for the derivative $D_1^i D_2^j g$.

2 Computational Complexity of Real Functions

This section reviews the complexity notions in Computable Analysis [9,16]. We start by fixing an encoding of real numbers by string functions.

---

time compubable functions g, such as $g(t, y) = y+1$, could yield functions h, such as $h(t) = \exp t - 1$, that grow too fast to be polynomial-time (or even polynomial-space) computable. Bournez, Graça and Pouly [1, Theorem 2] report that the statement about the analytic case holds true if we restrict the growth of h (and its extention to the complex plane) appropriately.

[6] The proof of this fact in [9, Theorem 3.7] needs to be fixed by redefining

$$f(x) = \begin{cases} u_s & \text{if not } R(s,t), \\ u_s + 2^{-(p(n)+2n+1)\cdot n} \cdot h_1(2^{p(n)+2n+1}(x - y_{s,t})) & \text{if } R(s,t). \end{cases}$$

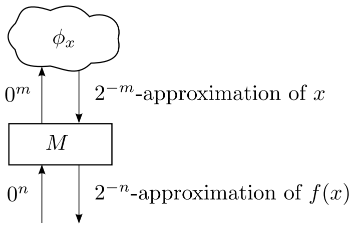

Fig. 1. A machine M computing a real function f.

Definition 3. *A function* $\phi\colon \{0\}^* \to \{0,1\}^*$ *is a* name *of a real number* x *if for all* $n \in \mathbf{N}$, $\phi(0^n)$ *is the binary representation of* $\lfloor x \cdot 2^n \rfloor$ *or* $\lceil x \cdot 2^n \rceil$, *where* $\lfloor \cdot \rfloor$ *and* $\lceil \cdot \rceil$ *mean rounding down and up to the nearest integer.*

In effect, a name of a real number x receives 0^n and returns an approximation of x with precision 2^{-n}.

We use *oracle Turing machines* (henceforth just *machines*) to work on these names (Figure 1). Let M be a machine and ϕ be a function from strings to strings. We write $M^\phi(0^n)$ for the output string when M is given ϕ as oracle and string 0^n as input. Thus we also regard M^ϕ as a function from strings to strings.

Definition 4. *Let* A *be a bounded closed interval of* $\mathbf{R}$. *A machine* M computes *a real function* $f\colon A \to \mathbf{R}$ *if for any* $x \in A$ *and any name* ϕ_x *of it,* M^{ϕ_x} *is a name of* $f(x)$.

Computation of a function $f\colon A \to \mathbf{R}$ on a two-dimensional bounded closed region $A \subseteq \mathbf{R}^2$ is defined in a similar way using machines with two oracles. A real function is (*polynomial-time*) *computable* if there exists some machine that computes it (in polynomial time). Polynomial-time computability of a real function f means that for any $n \in \mathbf{N}$, an approximation of $f(x)$ with error bound 2^{-n} is computable in time polynomial in n independent of the real number x.

By the time the machine outputs the approximation of $f(x)$ of precision 2^{-n}, it knows x only with some precision 2^{-m}, and this implies that all computable real functions are continuous.

To talk about hardness, we define reduction. A language $L \subseteq \{0,1\}^*$ is identified with the function $L\colon \{0,1\}^* \to \{0,1\}$ such that $L(u) = 1$ when $u \in L$.

Definition 5. *A language* L reduces *to a function* $f\colon [0,1] \to \mathbf{R}$ *if there exists a polynomial-time function* S *and a polynomial-time oracle Turing machine* M *(Figure 2) such that for any string* u,

(i) $S(u, \cdot)$ *is a name of a real number* x_u, *and*
(ii) $M^\phi(u)$ *accepts if and only if* $u \in L$ *for any name* ϕ *of* $f(x_u)$.

This definition may look stronger than the one in Kawamura [4], but is easily seen to have the same power. For a complexity class C, a function f is C*-hard* if all languages in C reduce to f.

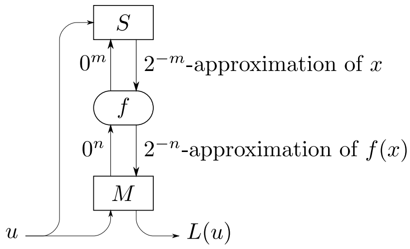

Fig. 2. Reduction from a language L to a function $f\colon [0,1] \to \mathbf{R}$

3 Proof of the Theorems

The proofs of Theorems 1 and 2 proceed as follows. In Section 3.1, we define *difference equations*, a discrete version of the differential equations. In Section 3.2, we show the PSPACE- and CH-hardness of difference equations with certain restrictions. In Section 3.3, we show that these classes of difference equations can be simulated by families of differential equations satisfying certain uniform bounds on higher-order derivatives. In Section 3.4, we prove the theorems by putting these families of functions together to obtain one differential equation having the desired smoothness ($\mathrm{C}^{(\infty,1)}$ and $\mathrm{C}^{(\infty,k)}$).

The idea of simulating a discrete system of limited feedback capability by differential equations was essentially already present in the proof of the Lipschitz version [4]. We look more closely at this limited feedback mechanism, and observe that this restriction is one on the *height* of the difference equation. We show that a stronger height restriction makes the difference equation simulable by smoother differential equations, leading to the CH-hardness for $\mathrm{C}^{(\infty,k)}$ functions.

3.1 Difference Equations

In this section, we define difference equations, a discrete version of differential equations, and show the PSPACE- and CH-hardness of families of difference equations with different height restrictions.

Let $[n]$ denote $\{0,\dots,n-1\}$. Let $G\colon [P]\times[Q]\times[R] \to \{-1,0,1\}$ and $H\colon [P+1]\times[Q+1] \to [R]$. We say that H is the solution of the *difference equation* given by G if for all $i \in [P]$ and $T \in [Q]$ (Figure 3),

$$H(i,0) = H(0,T) = 0, \tag{2}$$

$$H(i+1,T+1) - H(i+1,T) = G(i,T,H(i,T)). \tag{3}$$

We call P, Q and R the *height*, *width* and *cell size* of the difference equation. The equations (2) and (3) are similar to the initial condition $h(0) = 0$ and the equation $Dh(t) = g(t,h(t))$ in (1), respectively. In Section 3.3, we will simulate difference equations by differential equations using this similarity.

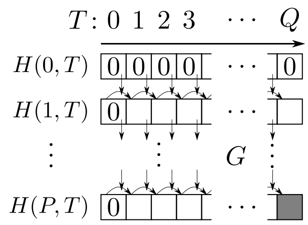

Fig. 3. The solution H of the difference equation given by G

We view a family of difference equations as a computing system by regarding the value of the bottom right cell (the gray cell in Figure 3) as the output. A family $(G_u)_u$ of functions $G_u\colon [P_u] \times [Q_u] \times [R_u] \to \{-1, 0, 1\}$ *recognizes* a language L if for each u, the difference equation given by G_u has a solution H_u and $H_u(P_u, Q_u) = L(u)$. A family $(G_u)_u$ is *uniform* if the height, width and cell size of G_u are polynomial-time computable from u (in particular, they must be bounded by $2^{p(|u|)}$, for some polynomial p) and $G_u(i, T, Y)$ is polynomial-time computable from (u, i, T, Y). A family $(G_u)_u$ has *polynomial height* if the height P_u is bounded by some polynomial $p(|u|)$. A family $(G_u)_u$ has *logarithmic height* if the height P_u is bounded by $c \log |u| + d$ with some constants c and d. With this terminology, the key lemma in [4, Lemma 4.7] can be written as follows:

Lemma 6. *There exists a* PSPACE*-hard language L that is recognized by some uniform family of functions with polynomial height*[7]*.*

Kawamura obtained the hardness result in the third row in Table 1 by simulating the difference equations of Lemma 6 by Lipschitz-continuous differential equations. Likewise, Theorem 1 follows from Lemma 6, by a modified construction that keeps the function in class $\mathrm{C}^{(\infty,1)}$ (Sections 3.3 and 3.4).

We show further that difference equations restricted to have logarithmic height can be simulated by $\mathrm{C}^{(\infty,k)}$ functions for each k (Sections 3.3 and 3.4). Theorem 2 follows from this simulation and the following lemma.

Lemma 7. *There exists a* CH*-hard language L that is recognized by some uniform family of functions with logarithmic height.*

The definition of the counting hierarchy CH, its connection to difference equations and the proof of Lemma 7 will be presented in Section 3.2.

[7] In fact, the languages recognized by uniform families with polynomial height coincide with PSPACE.

3.2 The Counting Hierarchy and Difference Equations of Logarithmic Height

The polynomial hierarchy PH is defined using non-deterministic polynomial-time oracle Turing machines:

$$\Sigma_0^p = \mathsf{P}, \qquad \Sigma_{n+1}^p = \mathsf{NP}^{\Sigma_n^p}, \qquad \mathsf{PH} = \bigcup_n \Sigma_n^p. \tag{4}$$

The counting hierarchy CH is defined similarly using probabilistic polynomial-time oracle Turing machines [14, 15]:

$$\mathsf{C}_0\mathsf{P} = \mathsf{P}, \qquad \mathsf{C}_{n+1}\mathsf{P} = \mathsf{PP}^{\mathsf{C}_n\mathsf{P}}, \qquad \mathsf{CH} = \bigcup_n \mathsf{C}_n\mathsf{P}. \tag{5}$$

It is known that $\mathsf{PH} \subseteq \mathsf{CH} \subseteq \mathsf{PSPACE}$, but we do not know whether $\mathsf{PH} = \mathsf{PSPACE}$.

Each level of the counting hierarchy has a complete problem defined as follows. For every formula $\phi(X)$ with the list X of l free propositional variables, we write

$$\mathsf{C}^m X\phi(X) \longleftrightarrow \sum_{X\in\{0,1\}^l} \phi(X) \geq m, \tag{6}$$

where $\phi(X)$ is identified with the function $\phi\colon \{0,1\}^l \to \{0,1\}$ such that $\phi(X) = 1$ when $\phi(X)$ is true. This "counting quantifier" C^m generalizes the usual quantifiers $\exists$ and $\forall$, because $\mathsf{C}^1 = \exists$ and $\mathsf{C}^{2^l} = \forall$. For lists $X_1, \ldots, X_n$ of variables and a formula $\phi(X_1, \ldots, X_n)$ with all free variables listed, we define

$$\langle\phi(X_1, \ldots, X_n), m_1, \ldots, m_n\rangle \in \mathsf{C}_n B_{be} \longleftrightarrow \mathsf{C}^{m_1} X_1 \cdots \mathsf{C}^{m_n} X_n \phi(X_1, \ldots, X_n). \tag{7}$$

Lemma 8 ([15, Theorem 7]). *For every $n \geq 1$, the problem $\mathsf{C}_n B_{be}$ is $\mathsf{C}_n\mathsf{P}$-complete.*

We define the problem $\mathsf{C}_{\log} B_{be}$ by

$$\langle 0^{2^n}, u\rangle \in \mathsf{C}_{\log} B_{be} \longleftrightarrow u \in \mathsf{C}_n B_{be}. \tag{8}$$

We show that $\mathsf{C}_{\log} B_{be}$ is CH-hard and recognized by a logarithmic-height uniform function family, as required in Lemma 7.

Proof (Lemma 7). First we prove that $\mathsf{C}_{\log} B_{be}$ is CH-hard. For each problem A in CH, there is a constant n such that $A \in \mathsf{C}_n\mathsf{P}$. From Lemma 8, for each $u \in \{0,1\}^*$ there is a polynomial-time function f_n such that $u \in A \leftrightarrow f_n(u) \in \mathsf{C}_n B_{be}$. So

$$u \in A \longleftrightarrow \langle 0^{2^n}, f_n(u)\rangle \in \mathsf{C}_{\log} B_{be}. \tag{9}$$

Since $\langle 0^{2^n}, f_n(\cdot)\rangle$ is polynomial time computable, A is reducible to $\mathsf{C}_{\log} B_{be}$.

Next we sketch (the details will be given in the full version of this paper) how to construct, for each formula with n counting quantifiers, a difference equation of height $n+1$ such that the output of its computation (i.e., the number in the bottom right cell) is equal to the value of the formula, and other cells in the last column contain 0. Once we have such a difference equation, we can construct a logarithmic-height uniform function family recognizing $\mathsf{C}_{\log}B_{be}$, since n is logarithmic in the length of the input of $\mathsf{C}_{\log}B_{be}$ because of the padding 0^{2^n} in (8). We build up the difference equation recursively. For $n=0$, it is easy because the value of a formula without quantifiers is polynomial-time computable. Consider the formula $\mathsf{C}^m X\psi(X)$, where $\psi(X)$ has i counting quantifiers, and assume that for each $Y \in \{0,1\}^l$, there exists a difference equation of height $i+1$ that computes $\psi(Y)$. By connecting these equations along the T-axis (i.e., arranging tables like Figure 3 horizontally), we construct a new difference equation that computes $\sum_Y \psi(Y)$ in the $(i+1)$st row and then uses this sum to compute $\mathsf{C}^m X\psi(X)$ in the next row. To bring the value in the $(i+1)$st row back to 0, we further connect the difference equation computing $-\sum_Y \psi(Y)$, which can be constructed similarly by changing the signs. $\square$

3.3 Families of Real Functions Simulating Difference Equations

We show that certain families of smooth differential equations can simulate PSPACE- or CH-hard difference equations stated in previous section.

Before stating Lemmas 9 and 10, we extend the definition of polynomial-time computability of real function to families of real functions. A machine M *computes* a family $(f_u)_u$ of functions $f_u\colon A \to \mathbf{R}$ indexed by strings u if for any $x \in A$ and any name ϕ_x of x, the function taking v to $M^{\phi_x}(u,v)$ is a name of $f_u(x)$. We say a family of real functions $(f_u)_u$ is polynomial-time if there is a polynomial-time machine computing $(f_u)_u$.

Lemma 9. *There exist a CH-hard language L and a polynomial μ, such that for any $k \geq 1$ and polynomials γ, there are a polynomial ρ and families $(g_u)_u$, $(h_u)_u$ of real functions such that $(g_u)_u$ is polynomial-time computable and for any string u:*

(i) $g_u\colon [0,1]\times[-1,1] \to \mathbf{R}$, $h_u\colon [0,1] \to [-1,1]$;
(ii) $h_u(0) = 0$ *and* $Dh_u(t) = g_u(t, h_u(t))$ *for all* $t \in [0,1]$;
(iii) g_u *is of class* $\mathrm{C}^{(\infty,k)}$;
(iv) $D^{(i,0)}g_u(0,y) = D^{(i,0)}g_u(1,y) = 0$ *for all* $i \in \mathbf{N}$ *and* $y \in [-1,1]$;
(v) $\left|D^{(i,j)}g_u(t,y)\right| \leq 2^{\mu(i,|u|)-\gamma(|u|)}$ *for all* $i \in \mathbf{N}$ *and* $j \in \{0,\dots,k\}$;
(vi) $h_u(1) = 2^{-\rho(|u|)}L(u)$.

Lemma 10. *There exist a PSPACE-hard language L and a polynomial μ, such that for any polynomial γ, there are a polynomial ρ and families $(g_u)_u$, $(h_u)_u$ of real functions such that $(g_u)_u$ is polynomial-time computable and for any string u satisfying (i)–(vi) of Lemma 9 with $k=1$.*

In Lemmas 9 and 10 we have the new conditions (iii)–(v) about the derivatives of g_u that were not present in [4, Lemma 4.1]. These conditions will permit the family of functions to be put together in one smooth function in Theorems 1 and 2.

We will prove Lemma 9 using Lemma 7 as follows. Let a function family $(G_u)_u$ be as in Lemma 7, and let $(H_u)_u$ be the family of the solutions of the difference equations given by $(G_u)_u$. We construct h_u and g_u from H_u and G_u such that $h_u(T/2^{q(|u|)}) = \sum_{i=0}^{p(|u|)} H_u(i,T)/B^{d_u(i)}$ for each $T = 0, \ldots, 2^{q(|u|)}$ and $Dh_u(t) = g_u(t, h_u(t))$. The polynomial-time computability of $(g_u)_u$ follows from that of $(G_u)_u$. We can prove Lemma 10 from Lemma 6 in the same way.

3.4 Proof of the Main Theorems

Using the function families $(g_u)_u$ and $(h_u)_u$ obtained from Lemmas 9 or 10, we construct the functions g and h in Theorems 1 and 2 as follows. Divide $[0, 1)$ into infinitely many subintervals $[l_u^-, l_u^+]$, with midpoints c_u. We construct h by putting a scaled copy of h_u onto $[l_u^-, c_u]$ and putting a horizontally reversed scaled copy of h_u onto $[c_u, l_u^+]$ so that $h(l_u^-) = 0$, $h(c_u) = 2^{-\rho'(|u|)}L(u)$ and $h(l_u^+) = 0$ where ρ' is a polynomial. In the same way, g is constructed from $(g_u)_u$ so that g and h satisfy (1). We give the details of the proof of Theorem 2 from Lemma 9, and omit the analogous proof of Theorem 1 from Lemma 10.

Proof (Theorem 2). Let L and μ be as Lemma 9. Define $\lambda(x) = 2x + 2$, $\gamma(x) = \mu(x, x) + x\lambda(x)$ and for each u let $\Lambda_u = 2^{\lambda(|u|)}$, $c_u = 1 - 2^{-|u|} + 2\bar{u} + 1/\Lambda_u$, $l_u^{\mp} = c_u \mp 1/\Lambda_u$, where $\bar{u} \in \{0, \ldots, 2^{|u|} - 1\}$ is the number represented by u in binary notation. Let ρ, $(g_u)_u$, $(h_u)_u$ be as in Lemma 9 corresponding to the above γ.

We define

$$g\left(l_u^{\mp} \pm \frac{t}{\Lambda_u}, \frac{y}{\Lambda_u}\right) = \begin{cases} \pm \sum_{l=0}^{k} \frac{D^{(0,l)} g_u(t,1)}{l!} (y-1)^l & \text{if } 1 < y, \\ \pm g_u(t,y) & \text{if } -1 \le y \le 1, \\ \pm \sum_{l=0}^{k} \frac{D^{(0,l)} g_u(t,-1)}{l!} (y+1)^l & \text{if } 1 < y, \end{cases} \tag{10}$$

$$h\left(l_u^{\mp} \pm \frac{t}{\Lambda_u}\right) = \frac{h_u(t)}{\Lambda_u} \tag{11}$$

for each string u and $t \in [0, 1)$, $y \in [-1, 1]$. Let $g(1, y) = 0$ and $h(1) = 0$ for any $y \in [-1, 1]$.

It can be shown similarly to the Lipschitz version [4, Theorem 3.2] that g and h satisfy (1) and g is polynomial-time computable. Here we only prove that g is of class $\mathrm{C}^{(\infty,k)}$. We claim that for each $i \in \mathbf{N}$ and $j \in \{0, \ldots, k\}$, the derivative $D_1^i D_2^j g$ is given by

$$D_1^i D_2^j g\left(l_u^{\mp} \pm \frac{t}{\Lambda_u}, \frac{y}{\Lambda_u}\right) = \begin{cases} \pm \Lambda_u^{i+j} \sum_{l=j}^{k} \frac{D^{(i,l)} g_u(t,1)}{(l-j)!} (y-1)^l & \text{if } y < -1, \\ \pm \Lambda_u^{i+j} D^{(i,j)} g_u(t,y) & \text{if } -1 \le y \le 1, \\ \pm \Lambda_u^{i+j} \sum_{l=j}^{k} \frac{D^{(i,l)} g_u(t,-1)}{(l-j)!} (y+1)^l & \text{if } 1 < y \end{cases} \tag{12}$$

for each $l_u^{\mp} \pm t/\Lambda_u \in [0,1)$ and $y/\Lambda_u \in [-1,1]$, and by $D_1^i D_2^j g(1,y) = 0$. This is verified by induction on $i+j$. The equation (12) follows from calculation (note that this means verifying that (12) follows from the definition of g when $i=j=0$; from the induction hypothesis about $D_2^{j-1} g$ when $i=0$ and $j>0$; and from the induction hypothesis about $D_1^{i-1} D_2^j g$ when $i>0$). That $D_1^i D_2^j g(1,y) = 0$ is immediate from the induction hypothesis if $i=0$. If $i>0$, the derivative $D_1^i D_2^j g(1,y)$ is by definition the limit

$$\lim_{s \to 1-0} \frac{D_1^{i-1} D_2^j g(1,y) - D_1^{i-1} D_2^j g(s,y)}{1-s}. \tag{13}$$

This can be shown to exist and equal 0, by observing that the first term in the numerator is 0 and the second term is bounded, when $s \in [l_u^-, l_u^+]$, by

$$\begin{aligned} |D_1^{i-1} D_2^j g(s,y)| &\le \Lambda_u^{i-1+j} \sum_{l=j}^{k} |D^{(i-1,l)} g_u| \cdot (\Lambda_u + 1)^l \\ &\le \Lambda_u^{i-1+j} \cdot k \cdot 2^{\mu(i-1,|u|)-\gamma(|u|)} \cdot (2\Lambda_u)^k \\ &\le 2^{(i-1+j+k)\lambda(|u|)+2k+\mu(i-1,|u|)-\gamma(|u|)} \le 2^{-2|u|} \le 2^{-|u|+1}(1-s), \end{aligned} \tag{14}$$

where the second inequality is from Lemma 9 (v) and the fourth inequality holds for sufficiently large $|u|$ by our choice of γ. The continuity of $D_1^i D_2^j g$ on $[0,1) \times [-1,1]$ follows from (12) and Lemma 9 (iv). The continuity on $\{1\} \times [-1,1]$ is verified by estimating $D_1^i D_2^j g$ similarly to (14). □

4 Complexity of Operators

Both Theorems 1 and 2 state the complexity of the solution h under the assumption that g is polynomial-time computable. But how hard is it to "solve" differential equations, i.e., how complex is the operator that takes g to h? To make this question precise, we need to define the complexity of operators taking real functions to real functions.

Recall that, to discuss complexity of real functions, we used string functions as names of elements in $\mathbf{R}$. Such an encoding is called a *representation* of $\mathbf{R}$. Likewise, we now want to encode real functions by string functions to discuss complexity of real operators. In other words, we need to define representations of the class $\mathrm{C}_{[0,1]}$ of continuous functions $h\colon [0,1] \to \mathbf{R}$ and class $\mathrm{CL}_{[0,1]\times[-1,1]}$ of Lipschitz continuous functions $g\colon [0,1] \times [-1,1] \to \mathbf{R}$. The notions of computability and complexity depend on these representations. Following [6], we use $\delta_\square$ and $\delta_{\square\mathrm{L}}$ as the representations of $\mathrm{C}_{[0,1]}$ and $\mathrm{CL}_{[0,1]\times[-1,1]}$, respectively. It is known that $\delta_\square$ is the canonical representation of $\mathrm{C}_{[0,1]}$ in a certain sense [5], and $\delta_{\square\mathrm{L}}$ is the representation defined by adding to $\delta_\square$ the information on the Lipschitz constant.

Since these representations use string functions whose values have variable lengths, we use *second order polynomials* to bound the amount of resources (time and space) of machines [6], and this leads to the definitions of second-order complexity classes (e.g. **FPSPACE**, polynomial-space computable), reductions (e.g. $\leq_{\mathrm{W}}$, polynomial-time Weihrauch reduction), and hardness. Combining them with the representations of real functions mentioned above, we can restate our theorems in the constructive form as follows.

Let *ODE* be the operator mapping a real function $g \in \mathrm{CL}_{[0,1]\times[-1,1]}$ to the solution $h \in \mathrm{C}_{[0,1]}$ of (1). The operator *ODE* is a partial function from $\mathrm{CL}_{[0,1]\times[-1,1]}$ to $\mathrm{C}_{[0,1]}$ (it is partial because the trajectory may fall out of the interval $[-1,1]$, see the paragraph following Theorem 2). In [6, Theorem 4.9], the $(\delta_{\square\mathrm{L}}, \delta_{\square})$-**FPSPACE**-$\leq_{\mathrm{W}}$-completeness of *ODE* was proved by modifying the proof of the results in the third row of Table 1. Theorem 1 can be rewritten in a similar way. That is, let ODE_k be the operator *ODE* whose input is restricted to class $\mathrm{C}^{(\infty,k)}$. Then we have:

Theorem 11. *The operator* ODE_1 *is* $(\delta_{\square\mathrm{L}}, \delta_{\square})$-**FPSPACE**-$\leq_{\mathrm{W}}$-*complete.*

To show this theorem, we need to verify that the information used to construct functions in the proof of Theorem 1 can be obtained easily from the inputs. We omit the proof since it does not require any new technique. Theorem 11 automatically implies Theorem 1 because of [6, Lemmas 3.7 and 3.8].

In contrast, the polynomial-time computability in the analytic case (the last row of Table 1) is *not* a consequence of a statement based on $\delta_{\square}$. It is based on the calculation of the Taylor coefficients, and hence we only know how to convert the Taylor sequence of g at a point to that of h. In other words, the operator *ODE* restricted to the analytic functions is not $(\delta_{\square\mathrm{L}}, \delta_{\square})$-**FP**-computable, but $(\delta_{\mathrm{Taylor}}, \delta_{\mathrm{Taylor}})$-**FP**-computable, where δ_{Taylor} is the representation that encodes an analytic function using its Taylor coefficients at a point in a suitable way. More discussion on the representation of analytic functions and the complexity of operators on them can be found in [7].

Acknowledgments

The writing of this paper was helped greatly by the comments and discussions at the 10th EATCS/LA Workshop on Theoretical Computer Science, held in Kyoto in early 2012, where the authors reported preliminary results. This work was supported in part by *Kakenhi* (Grant-in-Aid for Scientific Research) 23700009 and by the *Marie Curie International Research Staff Exchange Scheme Fellowship* 294962 within the 7th European Community Framework Programme.

References

1. O. Bournez, D. Graça, and A. Pouly. Solving analytic differential equations in polynomial time over unbounded domains. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011*, volume 6907 of *Lecture Notes in Computer Science*, pages 170–181. Springer, 2011.

2. E.A. Coddington and N. Levinson. *Theory of Ordinary Differential Equations*. McGraw-Hill, 1955.
3. A. Kawamura. Complexity of initial value problems. To appear in *Fields Institute Communications*.
4. A. Kawamura. Lipschitz continuous ordinary differential equations are polynomial-space complete. *Computational Complexity*, 19(2):305–332, 2010.
5. A. Kawamura. On function spaces and polynomial-time computability. Dagstuhl Seminar 11411: Computing with Infinite Data, 2011.
6. A. Kawamura and S. Cook. Complexity theory for operators in analysis. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, pages 495–502, 2010.
7. A. Kawamura, N. T. Müller, C. Rösnick, and M. Ziegler. Uniform polytime computable operators on univariate real analytic functions. In *Proceedings of the Ninth International Conference on Computability and Complexity in Analysis*, 2012.
8. K.I. Ko. On the computational complexity of ordinary differential equations. *Information and Control*, 58(1-3):157–194, 1983.
9. K.I. Ko. *Complexity Theory of Real Functions*. Birkhäuser Boston, 1991.
10. K.I. Ko and H. Friedman. Computing power series in polynomial time. *Advances in Applied Mathematics*, 9(1):40–50, 1988.
11. W. Miller. Recursive function theory and numerical analysis. *Journal of Computer and System Sciences*, 4(5):465–472, 1970.
12. N.T. Müller. Uniform computational complexity of Taylor series. *Automata, Languages and Programming*, pages 435–444, 1987.
13. M.B. Pour-el and I. Richards. A computable ordinary differential equation which possesses no computable solution. *Annals of Mathematical Logic*, 17(1-2):61–90, 1979.
14. J. Torán. Complexity classes defined by counting quantifiers. *Journal of the ACM*, 38(3):752–773, 1991.
15. K.W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23(3):325–356, 1986.
16. K. Weihrauch. *Computable Analysis: An Introduction*. Texts in Theoretical Computer Science. Springer, 2000.