

LIPSCHITZ CONTINUOUS INITIAL VALUE PROBLEM IS POLYNOMIAL-SPACE COMPLETE

AKITOSHI KAWAMURA

ABSTRACT. In answer to K. 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.

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

$$(1) \quad h(0) = 0, \quad h'(t) = g(t, h(t)), \quad t \in [0, 1].$$

The Picard–Lindelöf Theorem [16] states that this equation has a unique solution $h : [0, 1] \rightarrow \mathbf{R}$ if g is *Lipschitz continuous* (along its second argument), that is,

$$(2) \quad |g(t, y_0) - g(t, y_1)| \leq L \cdot |y_0 - y_1|, \quad t \in [0, 1], \quad y_0, y_1 \in \mathbf{R}$$

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

1. COMPUTATIONAL COMPLEXITY OF REAL FUNCTIONS

We discuss computation over the real numbers in the framework of *computable analysis* [8, 17], which will be reviewed briefly here. We assume that the reader is familiar with the basic concepts in complexity theory, including polynomial-time and polynomial-space machines, reductions, and completeness [13]. The computability notions of real numbers and of real functions discussed below date back at least to Turing [14] and Grzegorzczuk [3], respectively. Study on polynomial-time computability seems to have been started by Ko and Friedman [6].

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. Thus, $A(0^m)$ gives an approximation of t to precision 2^{-m} using a dyadic rational. We also say that A is a *name* of t . A real number is said to be (*polynomial-time*) *computable* if it has some (polynomial-time) computable name.

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 query state with a string v written on the

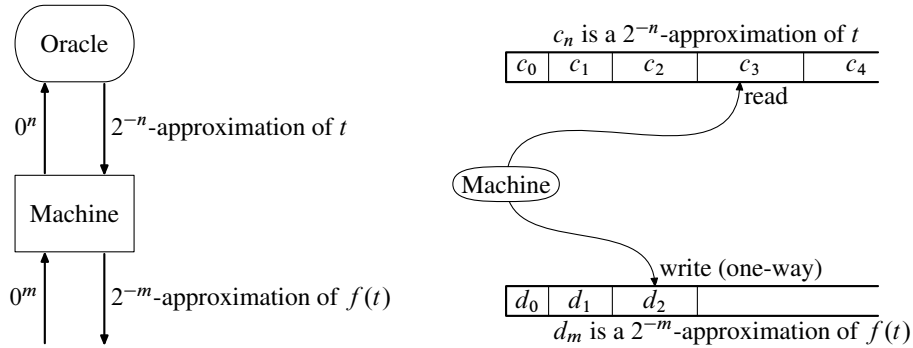


FIGURE 1. To compute a real function f , the machine should output an approximation of $f(t)$ to given precision 2^{-m} by consulting the oracle for approximations of t to any precision 2^{-n} it desires (left). An alternative picture (right) is that the machine converts any stream that encodes ever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

query tape; this string is then replaced by the answer $A(v)$ in one step. The string-to-string function computed by machine M with oracle A will be denoted by M^A .

Definition 1. A machine *computes* a function $f : [0, 1] \rightarrow \mathbf{R}$ 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 reduction of $f(t)$ to t (Figure 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 (Figure 1, right).

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

Many familiar continuous functions are computable. For example, it is not hard to see that the sine function restricted to $[0, 1]$ is polynomial-time computable, because an approximation of

$$(3) \quad \sin t = t - \frac{t^3}{3!} + \frac{t^5}{5!} - \frac{t^7}{7!} + \cdots$$

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 generalized to functions on compact intervals other than $[0, 1]$ and on two-dimensional rectangles (by considering the machine taking two oracles). Also, the definition of computation parametrized

by string u should be obvious. For example, a family $(g_u)_u$ of functions $g_u : [0, 1] \times [-1, 1] \rightarrow \mathbf{R}$ indexed by strings u is computed by a machine M if for any names A and B of t and y , 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 is polynomial-time means that it halts in time polynomial in $|u| + m$.

2. KO'S PROBLEM

We regard 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 \mathbf{P} , \mathbf{NP} , \mathbf{PSPACE} to denote the standard classes of languages [13]. We write $\mathbf{P}^{\#\mathbf{P}}$ is Valiant's counting class [15]. It is easy to see that $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{P}^{\#\mathbf{P}} \subseteq \mathbf{PSPACE}$, but none of the inclusions is known to be proper.

Friedman [1] studied whether it is the case that if $f : [0, 1] \rightarrow \mathbf{R}$ is polynomial-time computable, then so is $\text{MAX } f$ or $\text{INT } f$, where

$$(4) \quad (\text{MAX } f)(t) = \max_{\tau \in [0, t]} f(\tau), \quad (\text{INT } f)(t) = \int_0^t f(\tau) d\tau, \quad t \in [0, 1].$$

He showed that these questions are equivalent to $\mathbf{P} = \mathbf{NP}$ and $\mathbf{P} = \mathbf{P}^{\#\mathbf{P}}$, respectively. These results roughly correspond to the intuition that telling whether a number d is below the maximum of f is to test *existence* of a point at which f exceeds d , and that approximating the integral is to *count* the grid points below the graph of f .

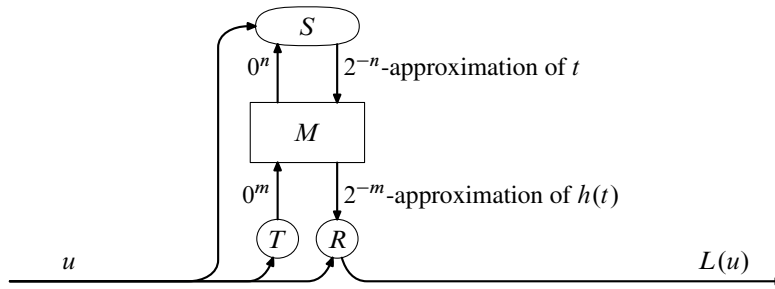
Ko considered the same question for, among many other operators [8, 10], our Lipschitz continuous initial value problem. He asked [7, Section 1] whether the following assumption (*) always implies that h is polynomial-time computable:

- (*) $g : [0, 1] \times \mathbf{R} \rightarrow \mathbf{R}$ satisfies (2); the values of $h : [0, 1] \rightarrow \mathbf{R}$ are always in $[-1, 1]$; functions g and h satisfy (1); and g restricted to $[0, 1] \times [-1, 1]$ is polynomial-time computable.

Since differential equations of form (1) subsume simple integration as a special case, Friedman's argument shows that (*) does not imply polynomial-time computability of h unless $\mathbf{P} = \mathbf{P}^{\#\mathbf{P}}$. Ko [7, Section 4] points out, through a careful analysis of the Euler method, that it would if $\mathbf{P} = \mathbf{PSPACE}$. He also shows [7, Section 5] that the other direction holds if Lipschitz continuity (2) is replaced by a weaker version. He conjectures that the exact complexity with the original Lipschitz condition lies somewhere strictly above $\mathbf{P}^{\#\mathbf{P}}$ [8, Section 7.4] and below \mathbf{PSPACE} [9, Section 1]. We will show that it is \mathbf{PSPACE} .

Theorem 2. *There are functions g and h with (*) such that h is \mathbf{PSPACE} -hard in the sense that for any $L \in \mathbf{PSPACE}$, there are polynomial-time computable functions R, S, T such that $L(u) = R(u, M^{S_u}(T(u)))$ for any string u and any machine M computing h (Figure 2), where S_u denotes the function that takes string 0^n to $S(u, 0^n)$.*

Corollary 3. $\mathbf{P} = \mathbf{PSPACE}$ if and only if (*) implies that h is polynomial-time computable.

FIGURE 2. R , S and T reduces L to M .

Proof. As mentioned above, Ko showed the forward direction [7, Section 4]. The other direction follows from Theorem 2. \square

Ko also 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. We answer it by the same technique. We write \mathbf{P}_1 and \mathbf{PSPACE}_1 to denote the classes of tally languages in \mathbf{P} and \mathbf{PSPACE} , respectively.

Theorem 4. *There are functions g and h with (*) such that $h(1)$ is \mathbf{PSPACE}_1 -hard in the sense that for any $L \in \mathbf{PSPACE}_1$, there are polynomial-time computable functions R and S such that $L(0^k) = R(A(S(0^k)))$ for any $k \in \mathbf{N}$ and any name A of t .*

Corollary 5. $\mathbf{P}_1 = \mathbf{PSPACE}_1$ if and only if (*) implies that $h(1)$ is polynomial-time computable.

We close this section with a few remarks to justify our formulation of the questions in these forms. The awkward restriction to $[-1, 1]$ in (*) could be avoided, without changing our construction below, by extending Definition 1 to functions with unbounded domain [8, pp. 57–58]. This would slightly complicate the definition of polynomial-time computability and also make it disagree with another formulation in literature [17, Chapter 7]. We just opted for simplicity of definition.

Our assumption of Lipschitz continuity is motivated by the following facts [11, 12] (modified by Ko to match our polynomial-time setting [7, Section 3]): if we simply remove the condition, the (no longer unique) solutions can be all non-computable; if we instead assume that the solution is unique, it is now computable but can require arbitrarily long time.

The practical concern may be closer to the complexity of the operator that takes g to h , rather than of the solution h itself. There is a canonical way to define computability (without time bound) of such operators [17, Chapter 6]. However, it is not obvious how to define polynomial-time computability. Ko proposes a definition but then argues that it is too restrictive for our purpose [8, Section 2.7].

3. PROOF OF THE THEOREMS

The first subsection reduces Theorems 2 and 4 to a common Lemma 6. The second subsection defines a discrete version of the initial value problem, shows its **PSPACE**-completeness, and uses it to prove Lemma 6.

3.1. Building blocks and how to put them together. We construct the functions g and h in Theorems 2 and 4 from a family of pairs of functions $(g_u)_u$ and $(h_u)_u$ as described in the following lemma. The function g is defined by arranging the reduced copies of g_u in the right pattern. The lemma will be proved later.

Lemma 6. *Let L be a language in **PSPACE** and let $\lambda : \mathbf{N} \rightarrow \mathbf{N}$ be a polynomial. Then there exist a polynomial $\rho : \mathbf{N} \rightarrow \mathbf{N}$ and families of functions $g_u : [0, 1] \times [-1, 1] \rightarrow \mathbf{R}$ and $h_u : [0, 1] \rightarrow [-1, 1]$ indexed by binary strings u such that the family $(g_u)_u$ is polynomial-time computable (in the sense explained at the end of Section 1) and the following hold for each u :*

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

Proof of Theorem 2. Divide $[0, 1]$ into infinitely many subintervals $[l_u, r_u]$, one for each u , with midpoints c_u . More precisely, for each $u = a_0 a_1 \dots a_{|u|-1}$, let

$$(5) \quad l_u = 1 - \frac{1}{2^{|u|}} + \frac{1}{2^{|u|+2}} \sum_{i=0}^{|u|-1} \frac{a_i}{2^i},$$

and $c_u - l_u = r_u - c_u = 1/4^{|u|+1}$. 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 $|g_u(t, y)| \leq 2^{\gamma(|u|)}$. Define

$$(6) \quad g \left(l_u + \frac{t}{4^{|u|+1}}, \frac{y}{4^{|u|+1} \cdot 2^{|u|+\gamma(|u|)}} \right) \\ = -g \left(r_u - \frac{t}{4^{|u|+1}}, \frac{y}{4^{|u|+1} \cdot 2^{|u|+\gamma(|u|)}} \right) = \frac{g_u(t, y)}{2^{|u|+\gamma(|u|)},$$

$$(7) \quad h \left(l_u + \frac{t}{4^{|u|+1}} \right) = h \left(r_u - \frac{t}{4^{|u|+1}} \right) = \frac{h_u(t)}{4^{|u|+1} \cdot 2^{|u|+\gamma(|u|)}$$

for each $t \in [0, 1]$ and $y \in \mathbf{R}$, and $g(1, y) = h(1) = 0$ for each $y \in \mathbf{R}$ (Figure 3).

This definition is consistent by (b) of Lemma 6. We show that these functions g and h satisfy (*). Equation (1) for $t \in [l_u, r_u]$ follows from the corresponding relation (c) of g_u and h_u , because we have reduced the abscissa, ordinate and slope with the right proportion in defining g and h . The Lipschitz condition (2) is satisfied with $L = 1$ by (d) and our choice of λ . To see that g is polynomial-time computable, note that the value (6) lies in $[-2^{-|u|}, 2^{-|u|}]$. When asked for an approximation of (6) to precision 2^{-m} , the machine can therefore safely answer 0 if $m < |u|$. Otherwise it can answer by computing $g_u(t, y)$ to the same precision,

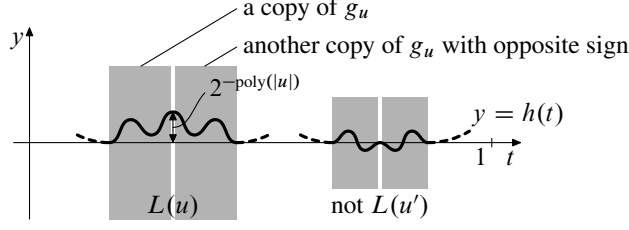


FIGURE 3. To construct g , we assign interval $[l_u, r_u]$ to each string u and put there a pair of reduced copies of g_u . The value $h(c_u)$ at the midpoint will be positive if and only if $L(u)$.

which can be done in time polynomial in $m + |u| \geq 2m$ by the polynomial-time computability of $(g_u)_u$. We have thus proved (*). Since

$$(8) \quad h(c_u) = \frac{L(u)}{4^{|u|+1} \cdot 2^{|u|+\gamma(|u|)} \cdot 2^{\rho(|u|)}}$$

by (7), it is routine to find the functions R, S, T in the statement. \square

Proof of Theorem 4. 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 $\gamma : \mathbf{N} \rightarrow \mathbf{N}$ satisfying $|g_{0^k}(t, y)| \leq 2^{\gamma(k)}$ for each k . Let $l_k = 1 - 2^{-k}$ and define

$$(9) \quad g\left(l_k + \frac{t}{2^{k+1}}, \frac{2j + (-1)^j y}{2^{2k+\gamma(k)+\bar{\rho}(k)}}\right) = \frac{g_{0^k}(t, y)}{2^{k-1+\gamma(k)+\bar{\rho}(k)}},$$

$$(10) \quad h\left(l_k + \frac{t}{2^{k+1}}\right) = \sum_{\kappa=0}^{k-1} \frac{L(0^\kappa)}{2^{2\kappa+\gamma(\kappa)+\bar{\rho}(\kappa)+\rho(\kappa)}} + \frac{h_{0^k}(t)}{2^{2k+\gamma(k)+\bar{\rho}(k)}}$$

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

$$(11) \quad h(1) = \sum_{k=0}^{\infty} \frac{L(0^k)}{2^{2k+\gamma(k)+\bar{\rho}(k)+\rho(k)}}.$$

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 2. Polynomial-time computability also follows from that of $(g_u)_u$ again. Since all summands on the right-hand side in (10) but the last are divisible by $4/2^{2k+\gamma(k)+\bar{\rho}(k)}$, substituting (10) into the second argument of (9) yields

$$(12) \quad g\left(l_k + \frac{t}{2^{k+1}}, h\left(l_k + \frac{t}{2^{k+1}}\right)\right) = \frac{g_{0^k}(t, h_{0^k}(t))}{2^{k-1+\gamma(k)+\bar{\rho}(k)}}.$$

This shows (1) for each $t \in [l_k, l_{k+1}]$ by (c). We have thus proved (*). By (11), it is routine to find R and S in the statement. \square

3.2. The discrete initial value problem. An attempt to prove Lemma 6 would be as follows. Consider a polynomial-space Turing machine that decides if 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. Thus, there is a simple rule that maps u (the input), T (time) and d (the previous configuration) to a number $G_u(T, d)$ (the next configuration) such that the recurrence

$$(13) \quad H_u(0) = 0, \quad H_u(T + 1) = G_u(T, H_u(T))$$

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 $L(u)$. We may be tempted to try to simulate the “discrete initial value problem” (13) by embedding each value N into real number $N/2^{C(|u|)}$.

The obstacle to this attempt is that the differential equation of Lemma 6 cannot express all discrete recurrences (13): trajectories of the (continuous) initial value problem cannot branch or cross one another, and besides, we have the Lipschitz condition. To get around this obstacle, we will now restrict the discrete problem (13) so that it can be simulated by differential equations and yet its final value has the complexity of **PSPACE**. Let C , P and Q be polynomials and let

$$(14) \quad G_u : [2^{Q(|u|)} + 1] \times [2^{C(|u|)}] \rightarrow \mathbf{Z}, \quad j_u : [2^{Q(|u|)} + 1] \rightarrow [P(|u|)],$$

$$(15) \quad H_u : [P(|u|)] \times [2^{Q(|u|)} + 1] \rightarrow [2^{C(|u|)}],$$

where we write $[N] = \{0, \dots, N - 1\}$ for $N \in \mathbf{N}$. Our restricted discrete initial value problem is as follows (Figure 4):

$$(16) \quad H_u(i, 0) = 0,$$

$$(17) \quad H_u(i, T + 1) = \begin{cases} H_u(i, T) + G_u(T, H_u(i - 1, T)) & \text{if } i = j_u(T) > 0, \\ H_u(i, T) & \text{otherwise.} \end{cases}$$

In effect, $H_u(T)$ of (13) is now divided into components $H_u(0, T), H_u(1, T), \dots$. We have added the restriction that G_u sees only the component $H_u(i - 1, T)$, which in the figure means the upper left of the current cell. 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 be a language in **PSPACE**. Then there are polynomials C , P , Q and polynomial-time computable families $(G_u)_u$, $(j_u)_u$ and $(H_u)_u$, indexed by strings u , that for each u satisfy (14)–(17) and*

$$(18) \quad H_u(i, 2^{Q(|u|)}) = \begin{cases} L(u) & \text{if } i = P(|u|) - 1, \\ 0 & \text{otherwise.} \end{cases}$$

We could have allowed G to see all of the upper components $H_u(0, T), \dots, H_u(i - 1, T)$; restriction to the last one is just for simplicity. Also inessential is the requirement in (18) that all but the last components of the final value be 0: if they

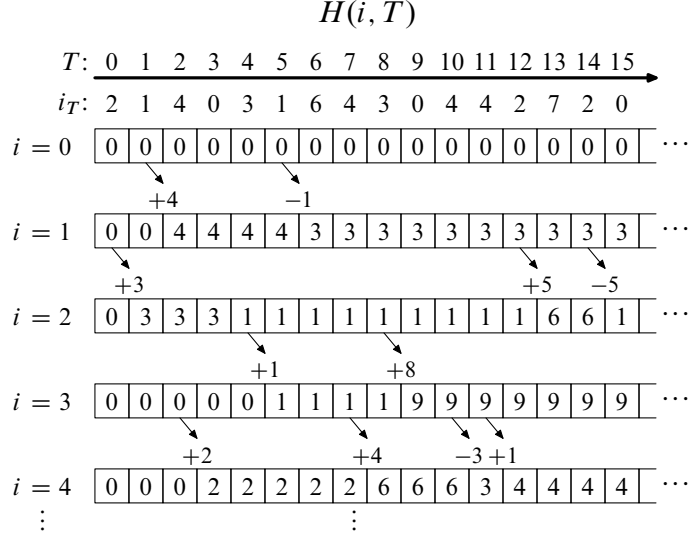


FIGURE 4. The column $(H_u(0, T), H_u(1, T), \dots)$ holds the state at each time T . The index $j_u(T)$ tells which row to increment next (0 means no increment). As the arrows indicate, G_u computes the increment from the upper left cell $H_u(j_u(T) - 1, T)$.

are not, they can be brought to 0 in twice as much time by defining G symmetrically to cancel out what it has done.

Proof of Lemma 7. We may assume that L is the problem that asks the truth value $L(u)$ of given formula u of form

$$(19) \quad Q_n x_n \dots Q_1 x_1 \cdot \psi(x_1, \dots, x_n),$$

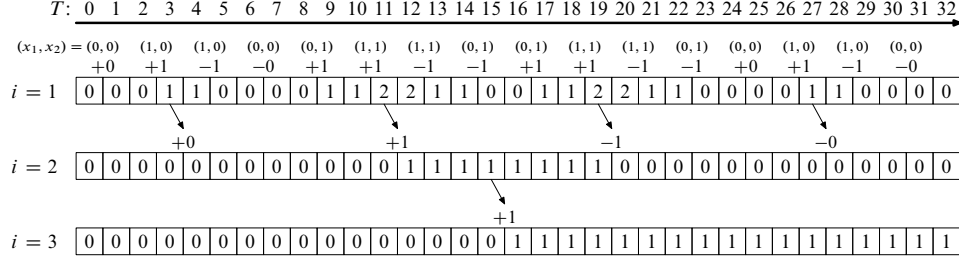
where ψ is a (quantifier-free) propositional formula and $Q_1, \dots, Q_n \in \{\forall, \exists\}$. This is because this problem, commonly called QBF or TQBF, is **PSPACE**-complete [13].

Write T_i for the i^{th} bit of integer T in binary: $T = \sum_i 2^i T_i$. Let $j_u(T)$ be the positive integer, if any, such that $T + 1 = 4^{j_u(T)-1} b$ for some odd number b ; otherwise, let $j_u(T) = 0$. Define G_u by

$$(20) \quad G_u(T, Y) = (-1)^{T_{2j_u(T)}} \times \begin{cases} \psi(T_1 \oplus T_2, T_3 \oplus T_4, \dots, T_{2n-1} \oplus T_{2n}) & \text{if } j_u(T) = 1, \\ 1 & \text{if } j_u(T) > 1 \text{ and } (Q_{j_u(T)-1}, Y) \in \{(\exists, 1), (\exists, 2), (\forall, 2)\}, \\ 0 & \text{if } j_u(T) > 1 \text{ and } (Q_{j_u(T)-1}, Y) \in \{(\exists, 0), (\forall, 0), (\forall, 1)\}. \end{cases}$$

We claim that $H_u(n + 1, 2^{2n+1}) = L(u)$ and $H_u(i, 2^{2n+1}) = 0$ for all other $i \neq n + 1$. Once we have shown this, we can achieve (18) of Lemma 7 for some fixed P and Q by adding some dummy rows and columns, because $n < |u|$.

We explain this by example, omitting the routine general proof. Let u for now be the formula $\exists x_2. \forall x_1. (x_1 \vee x_2)$ (Figure 5). The increments for the row $i = 1$

FIGURE 5. A discrete initial value problem evaluates formula $\exists x_2. \forall x_1. (x_1 \vee x_2)$.

encode (somewhat redundantly) the truth table of the matrix $x_1 \vee x_2$, as described in the first case of (20). For example, $G_u(2, 0)$, $G_u(4, 0)$, $G_u(26, 0)$ and $G_u(28, 0)$ are ± 1 because $(x_1, x_2) = (1, 0)$ makes $x_1 \vee x_2$ true, whereas $G_u(0, 0)$, $G_u(6, 0)$, $G_u(24, 0)$ and $G_u(30, 0)$ are 0 because $(x_1, x_2) = (0, 0)$ does not. Also observe that the coefficient in (20) makes $H_u(1, T)$ return to 0 every eight cells. As a result, the cell $H_u(1, 3) = 1$ (resp. $H_u(1, 11) = 2$) represents the fact that when x_2 is set to false (resp. true), formula $x_1 \vee x_2$ is satisfied by one (resp. two) of the assignments to x_1 .

Now look at the next row $i = 2$. The second and third cases of (20) says that this row gets incremented only when $T + 1$ is an odd multiple of 4, and that, since $Q_1 = \forall$ in our example, the increment is ± 1 or ± 0 according to whether the upper left cell has a 2 or not. As a result, the increments for this row encode, in the same way as the previous row, the smaller truth table for the subformula $\forall x_1. (x_1 \vee x_2)$ under each assignment to x_2 . The cell $H_u(2, 15) = 1$ represents the fact that this subformula is satisfied by one of the assignments to x_2 , which is why the last row gets incremented at time $T = 16$. We observe that our final cell $H_u(3, 32)$ has a 1 precisely because the whole formula u is true. \square

Now we prove Lemma 6 from Lemma 7 by simulating the discrete initial value problem (14)–(17) by a continuous trajectory.

Proof of Lemma 6. Take C , P , Q , $(G_u)_u$, $(j_u)_u$, $(H_u)_u$ from Lemma 7. Let $B = 2^{\lambda(|u|)+C(|u|)+Q(|u|)+3}$. Define $g_u : [0, 1] \times \mathbf{R} \rightarrow \mathbf{R}$ as follows: let

$$(21) \quad g_u(t, y) = \frac{2^{Q(|u|)} \pi \sin(\theta \pi)}{2^{B^{j_u(T)+1}}} \cdot G_u(T, Y),$$

when $\eta \in [0, 1/2]$, where

$$(22) \quad 2^{Q(|u|)} t = T + \theta, \quad T \in [2^{Q(|u|)}], \theta \in [0, 1],$$

$$(23) \quad B^{j_u(T)} y = 2^{C(|u|)} N + Y + \eta, \quad N \in \mathbf{N}, Y \in [2^{C(|u|)}], \eta \in [0, 1];$$

when $\eta \in (1/2, 1)$, define $g_u(t, y)$ by interpolating linearly in y (for each fixed t). Define $h_u : [0, 1] \rightarrow \mathbf{R}$ by

$$(24) \quad h_u(t) = \frac{1 - \cos(\theta\pi)}{2} \cdot \frac{H(j_u(T), T + 1) - H(j_u(T), T)}{B^{j_u(T)+1}} + \sum_{i=0}^{\infty} \frac{H_u(i, T)}{B^{i+1}},$$

where again (22). It is easy to verify that we have indeed defined continuous functions, that they satisfy (c) of Lemma 6 because of the discrete relation (16) and (17), and that the family $(G_u)_u$ is polynomial-time computable. Conditions (a), (b) and (e) are also obvious, with $\rho(k) = (P(k) + 1)(\lambda(k) + C(k) + Q(k) + 3)$. To see the Lipschitz condition (d), note that $g(t, y)$ can never change faster with respect to y than when, as η runs through the interval $(1/2, 1)$, the value $G(T, Y)$ switches between $\pm 2^{C(|u|)}$, thus changing (21) by $2^{Q(|u|)+C(|u|)}\pi \sin(\theta\pi)/B^{j_u(T)+1}$. The slope in this situation is

$$(25) \quad 2B^{j_u(T)} \cdot \frac{2^{Q(|u|)+C(|u|)}\pi \sin(\theta\pi)}{B^{j_u(T)+1}} \leq 2 \cdot \frac{2^{Q(|u|)+C(|u|)} \cdot 4 \cdot 1}{B} = 2^{-\lambda(|u|)}$$

by our choice of B . □

4. RELATED WORKS

Ko later considered [9] two classes of equations more general than (1). The complexity of *Volterra integral equations of the first kind* (with certain conditions) is shown to be between exponential time and exponential space. He also studies *Volterra integral equations of the second kind*

$$(26) \quad u(y) = f(y) + \int_0^y K(y, s, u(s)) ds, \quad y \in [0, 1],$$

where function u is to be solved from given functions K and f . For this equation, he shows (under the Lipschitz condition) the upper bound of polynomial space by analyzing Picard's successive approximation. Since (26) contains our initial value problem (1) as a special case, we have closed the complexity gap also for this class of equations.

If the function g satisfies the Lipschitz condition only locally, the solution h of (1) may diverge to infinity. Positive [5] and negative [2] results are known for the computability of the domain of h in this case (in certain precise senses of computability of real sets).

It is interesting to note that if we further assume that g is analytic, then h is polynomial-time computable [4].

Acknowledgement. (To be added in the published version.)

REFERENCES

- [1] H. Friedman. The computational complexity of maximization and integration. *Advances in Mathematics*, 53:80–98, 1984.

- [2] 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*, pages 127–135, 2007.
- [3] A. Grzegorzcyk. Computable functionals. *Fundamenta Mathematicae*, 42:168–202, 1955.
- [4] A. Kawamura. Complexity of initial value problems. Submitted.
- [5] A. Kawamura. Differential recursion. *ACM Transactions on Computational Logic*, 10, 2009. To appear.
- [6] K. Ko and H. Friedman. Computational complexity of real functions. *Theoretical Computer Science*, 20(3):323–352, 1982.
- [7] K.-I. Ko. On the computational complexity of ordinary differential equations. *Information and Control*, 58:157–194, 1983.
- [8] K.-I. Ko. *Computational Complexity of Real Functions*. Birkhäuser Boston, 1991.
- [9] K.-I. Ko. On the computational complexity of integral equations. *Annals of Pure and Applied Logic*, 58(3):201–228, November 1992.
- [10] K.-I. Ko. Polynomial-time computability in analysis. In I. L. Ershov et al., editors, *Handbook of Recursive Mathematics: Volume 2: Recursive Algebra, Analysis and Combinatorics*, volume 139 of *Studies in Logic and the Foundations of Mathematics*, pages 1271–1317. North-Holland, December 1998.
- [11] W. Miller. Recursive function theory and numerical analysis. *Journal of Computer and System Sciences*, 4:465–472, 1970.
- [12] 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.
- [13] M. Sipser. *Introduction to the Theory of Computation*. Course Technology, second edition, 2005.
- [14] A. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society (Series 2)*, 42:230–265, 1937.
- [15] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.
- [16] W. Walter. *Ordinary differential equations*, volume 182 of *Graduate Texts in Mathematics, Readings in Mathematics*. Springer-Verlag, New York, 1998. Translated by Russell Thompson from the sixth edition of *Gewöhnliche Differentialgleichungen*, 1996.
- [17] K. Weihrauch. *Computable Analysis: An Introduction*. Texts in Theoretical Computer Science. Springer-Verlag, 2000.