INFORMATION AND CONTROL 58, 157–194 (1983)

On the Computational Complexity of Ordinary Differential Equations*

KER-I KO

Department of Computer Science, University of Houston, Houston, Texas 77004

The computational complexity of the solution y of the differential equation $y'(x) = f(x, y(x))$, with the initial value $y(0) = 0$, relative to the computational complexity of the function f is investigated. The Lipschitz condition on the function f is shown to play an important role in this problem. On the one hand, examples are given in which f is polynomial time computable but none of the solutions y is computable. On the other hand, if f is polynomial time computable and if f satisfies a weak form of the Lipschitz condition then the (unique) solution y is polynomial space computable. Furthermore, there exists a polynomial time computable function f which satisfies this weak Lipschitz condition such that the (unique) solution y is not polynomial time computable unless $P = \text{PSPACE}$.

1. INTRODUCTION

Let $f(x, y)$ be a continuous function of 2 real variables on the rectangle $[0, 1] \times [-1, 1]$. Consider the following ordinary differential equation with initial condition:

$$y'(x) = f(x, y(x)), \qquad y(0) = 0. \tag{1}$$

The question to be studied in this paper is the following: What is the computational complexity of the solution y of Eq. (1) relative to that of function f? This question has been investigated by Henrici (1962), Cleave (1969), Aberth (1971), Miller (1970), and Pour-El and Richards (1979) at the recursive level and at the higher complexity level. Their results can be summarized as follows.

(1) If the function f is computable (in the formal sense of recursiveness in recursion theory) and Eq. (1) has a unique solution y, then y is also computable (Pour-El and Richards, 1979).

(2) There exists a function $f(x, y)$ computable on the rectangle

* This research was supported in part by the National Science Foundation under Grant MCS-8103479.

0019-9958/83 $3.00

$[0, 1] \times [-1, 1]$ but no solution of Eq. (1) is computable on any interval $[0, \delta]$, $\delta > 0$ (Aberth, 1971; Pour-El and Richards, 1979).

(3) For any recursive real number $a \in (0, 1)$, there exists a primitive recursive real function f on $[0, 1] \times [-1, 1]$ such tht $y(x) = ax^2$ is the unique solution of Eq. (1) (Miller, 1970).

(4) Assume that f satisfies the Lipschitz condition (i.e., there is a constant $L > 0$ such that $|f(x, z_1) - f(x, z_2)| \leqslant L \cdot |z_1 - z_2|$ for all $x \in [0, 1]$ and $z_1, z_2 \in [-1, 1]$). Then, for any $n \geqslant 3$, the (unique) solution y of Eq. (1) is in $\mathscr{E}^{(n)}$, the nth level of the Grzegorczyk hierarchy (Grzegorczyk, 1953), whenever $f \in \mathscr{E}^{(n)}$ (Cleave, 1969).

In this paper we restrict the function f to be polynomial time computable (see Sect. 2 for the definition), and ask the following questions.

Question A. Assume that f is polynomial time computable. Does Eq. (1) have a polynomial time computable solution?

Our answer to Question A is a negative one. Actually, we will show that a simple modification of Pour-El and Richards's proof of result (2) results in the following theorem.

THEOREM 1. *There exists a polynomial time computable function f on $[0, 1] \times [-1, 1]$ such that Eq. (1) has no computable solution y on $[0, \delta]$ for any $\delta > 0$.*

Of course, Eq. (1), with respect to the above function f, does not have a unique solution as indicated in result (1). What happens if we know that the equation has a unique solution?

Question B. Assume that f is polynomial time computable and Eq. (1) has a unique solution y on $[0, 1]$. Is y a polynomial time computable function?

Again, a simple modification of Pour-El and Richards's proof of result (2) (or, that of Miller's proof of result (3)) gives a negative answer to Question B.

THEOREM 2. *For any recursive function ϕ, there is a polynomial time computable real function f on $[0, 1] \times [-1, 1]$ such that* Eq. (1) *has a unique solution y on $[0, 1]$ but y is not computable by any oracle Turing machine* (TM) *operating in time ϕ.*

(For the time complexity of an oracle TM and its relation to the compuation of a real function, see Sect. 2.)

From the above two theorems, we know that the time complexity constraint only on the function f is not sufficient to control the time complexity of the solution y. So, following Cleave's observation, we ask

Question C. Assume that f is polynomial time computable and f satisfies the Lipschitz condition. Is the (unique) solution y of Eq. (1) polynomial time computable?

From Henrici's analysis of Euler's method (Henrici, 1962), as well as Cleave's result (4), it is clear that the solution y of Eq. (1) is polynomial *space* computable if f satisfies the conditions of Question C. Therefore, a negative answer to Question C would imply the existence of a polynomial space computable function y which is not polynomial time computable, and would solve negatively the famous $P = ?\text{PSPACE}$ question in discrete complexity theory. (We let P and PSPACE represent all sets contained in $\{0, 1\}^*$ which are computable in polynomial time and polynomial space, respectively. Also let P_f and PSPACE_f represent all functions ϕ: $\{0, 1\}^* \to \{0, 1\}^*$ computable in polynomial time and polynomial space, respectively. Note that $P = \text{PSPACE}$ if and only if $P_f = \text{PSPACE}_f$. See Garey and Johnson (1979) for more discussions on the question $P = ?\text{PSPACE}$.)

On the other hand, Friedman (1984) has shown that if $P_f \neq \#P_f$ then there exists a polynomial time computable real function g on $[0, 1]$ whose integral function $h(x) = \int_0^x g(t)\,dt$ is not polynomial time computable. ($\#P_f$ represents all functions which compute the number of accepting computations of polynomial time nondeterministic Turing machines. It is known that $P_f \subseteq \#P_f \subseteq \text{PSPACE}_f$, but $P_f = ?\#P_f$ and $\#P_f = ?\text{PSPACE}_f$ are unknown (Garey and Johnson, 1979).) Since integration of a 1-variable function is a subproblem of the question of finding solutions of Eq. (1), an affirmative answer to Question C would imply $P_f = \#P_f$.

In other words, we have [$P_f = \text{PSPACE}_f \Rightarrow$ Question C has an affirmative answer $\Rightarrow P_f = \#P_f$]. In view of the difficulty of resolving these two major open questions in discrete complexity theory, we ask, more realistically, the following weaker question.

Question D. Does there exist a natural complexity class F between $\#P_f$ and PSPACE_f such that Question C has an affirmative answer if and only if $P_f = F$?

If we examine Cleave's result (4) more closely, we can see that the Lipschitz condition on the function f is too strong and is not a necessary condition for the polynomial space computability of the solution y of Eq. (1). By replacing the Lipschitz condition by a weaker condition, we are able to answer a variant of Question D with $F = \text{PSPACE}_f$.

DEFINITION 1. We say a real function f on $[0, 1] \times [-1, 1]$ satisfies the *right Lipschitz condition* on a set $E \subseteq [0, 1] \times [-1, 1]$ if there exists a

constant $L > 0$ such that for all $x \in [0, 1]$ and $z_1, z_2 \in [-1, 1]$, if $z_1 < z_2$ and the line segment connecting (x, z_1) and (x, z_2) lies entirely inside E, then

$$f(x, z_2) - f(x, z_1) \leqslant L \cdot (z_2 - z_1).$$

Our main result is the following.

THEOREM 3. *The following statements are equivalent.*

(a) *Let f be a polynomial time computable function on $[0, 1] \times [-1, 1]$ such that Eq. (1) with respect to f has a unique solution y on $[0, 1]$. Assume that there is a set $E \subseteq [0, 1] \times [-1, 1]$, and a polynomial function α, such that*

(i) *f satisfies the right Lipschitz condition on E, and*

(ii) *for each $k \geqslant 1$, $\{(x, z)\colon 0 \leqslant x \leqslant 1 - 2^{-k}, |z - y(x)| \leqslant 2^{-\alpha(k)}\} \subseteq E$.*

Then, y is polynomial time computable.

(b) $P = \mathrm{PSPACE}$.

The proof technique for the direction (b) $\Rightarrow$ (a) is simply an error analysis for Euler's method, similar to the one given by Henrici (1962). For the direction (a) $\Rightarrow$ (b), we use the PSPACE-completeness of the quantified Boolean formula problem (QBF) (Garey and Johnson, 1979). We show that there exists a function f satisfying the conditions in Theorem 3(a) such that the QBF problem is reducible to the function y in polynomial time. One of the ideas in the proof is alpha–beta pruning (Knuth and Moore, 1975) of a self-reducing tree of a Boolean formula. (The self-reduciblity of Boolean formulas has been noticed by many people, e.g., Meyer and Paterson (1979), Schnorr (1976), and Ko (1983). Our definition is close to that of Meyer and Paterson (1979), and will be given in Sect. 5.2.)

The same proof technique also yields a characterization of the value $y(1)$ of Eq. (1). Let P_s and PSPACE_s denote the class of sets in $\{0\}^*$ which are in P and PSPACE, respectively. It is clear that $P = \mathrm{PSPACE} \Rightarrow P_s = \mathrm{PSPACE}_s$. Whether the converse holds is not known (cf. Book, 1974).

THEOREM 4. *The following statements are equivalent.*

(a) *Let f be a function defined on $[0, 1] \times [-1, 1]$ which satisfies the conditions in Theorem 3(a). Then, $y(1)$ is a polynomial time computable real number.*

(b) $P_s = \mathrm{PSPACE}_s$.

In addition, we observe that our proof for (b) $\Rightarrow$ (a) of Theorem 3 actually shows that for any function f which satisfies the conditions in Theorem 3(a),

its corresponding solution y of Eq. (1) can be computed by Euler's method in polynomial space. Therefore, we have the following corollary.

COROLLARY 1. *The following statements are equivalent.*

(a) *Let f be a polynomial time computable function on $[0, 1] \times [-1, 1]$. Assume that the solution y of Eq. (1) with respect to f is unique and can be approximated by Euler's method in a polynomial speed (in the sense that for some polynomial function α, the solution $y_{\alpha(n)}$ constructed by Euler's method with increment $2^{-\alpha(n)}$ is close to y within an error $\leqslant 2^{-n}$). Then y is polynomial time computable.*

(b) $P = \text{PSPACE}$.

One of the implications of Corollary 1 is that for some type of functions f, Euler's method for Eq. (1) operates at least as efficiently as any other method (within a polynomial factor), assuming $P \neq \text{PSPACE}$.

The plan for the paper is as follows. We review in Section 2 the basic definitions and properties of computational complexity of real functions and the theory of discrete complexity. In Section 3, we sketch the proofs of Theorems 1 and 2. The ideas of these proofs are from Pour-El and Richards's proof of result (2). The proofs of direction (b) $\Rightarrow$ (a) of Theorems 3 and 4 are given in Section 4. They are basically an error analysis for Euler's method. In Section 5, we prove our main results, direction (a) $\Rightarrow$ (b) of Theorems 3 and 4. The proofs are quite involved. So, we give a sketch in Section 5.1. Section 5.2 contains a basic construction for both theorems. A discrete version of the initial value problem is first constructed. Then a conversion to the continuous version follows. Details of the proofs of (a) $\Rightarrow$ (b) of Theorems 3 and 4 are in Sections 5.3 and 5.4, respectively.

The function f constructed in Section 5.3 does not satisfy the Lipschitz condition on $[0, 1] \times [-1, 1]$. Therefore, the original Question D remains open. Can we replace our weak, local Lipschitz condition in Theorem 3 by the simple Lipschitz condition? Or, is the problem of solving an ordinary differential equation with respect to a function satisfying the Lipschitz condition no more difficult than the integration problem? We feel that these questions are interesting to both numerical analysts and complexity theorists.

2. PRELIMINARIES

In this section we review some basic notions of computational complexity and give the basic definitions and properties of polynomial time and polynomial space computable real functions. The reader is referred to Garey and Johnson (1979) and Ko and Friedman (1982) for details.

We assume that the reader is familiar with Turing machines (TMs) and oracle Turing machines (oracle TMs). We say a TM M has *time complexity* (or, *space complexity*) $\leqslant \alpha$, an integer function, if for any input binary string s of length n, $M(s)$ halts in $\alpha(n)$ moves (or, halts and uses $\leqslant \alpha(n)$ cells, respectively). We say an oracle TM M has *time complexity* (or, *space complexity*) $\leqslant \alpha$, if for any input binary string s of length n and for any oracle function ϕ (an oracle set A can be represented by its characteristic function χ_A), $M^{\phi}(s)$ halts in $\leqslant \alpha(n)$ moves (or, halts and uses $\leqslant \alpha(n)$ cells, respectively). The class P_f (PSPACE_f) is the class of functions ϕ: $\{0,1\}^* \to \{0,1\}^*$ computable by a TM with polynomial time (or, space) complexity. The class P (PSPACE) is the class of languages $A \subseteq \{0,1\}^*$ such that $\chi_A \in P_f$ (or, $\chi_A \in \mathrm{PSPACE}_f$, respectively). It is clear that $P_f = \mathrm{PSPACE}_f$ implies $P = \mathrm{PSPACE}$. Conversely, for any function ϕ: $\{0,1\}^* \to \{0,1\}^*$, define $S_\phi = \{\langle s, t\rangle : t \leqslant \phi(s)\}$, where $\leqslant$ is the lexicographic order on $\{0,1\}^*$. Then clearly $\phi \in \mathrm{PSPACE}_f$ iff $S_\phi \in \mathrm{PSPACE}$ and $\phi \in P_f$ iff $S_\phi \in P$. So, $P = \mathrm{PSPACE}$ iff $P_f = \mathrm{PSPACE}_f$. The class P_{sf} (PSPACE_{sf}) denotes the class of functions ϕ: $\{0\}^* \to \{0,1\}^*$ which are in P_f (in PSPACE_f, respectively). It is clear that $P_f = \mathrm{PSPACE}_f$ implies $P_{sf} = \mathrm{PSPACE}_{sf}$. It is not known whether the converse holds. Also let P_s (PSPACE_s) be the class of languages $A \subseteq \{0\}^*$ which are in P (PSPACE, respectively).

A set $A \subseteq \{0,1\}^*$ is said to be *PSPACE-complete* (with respect to the $\leqslant_m^P$-reduction) if (i) $A \in \mathrm{PSPACE}$ and (ii) for any set B in PSPACE there exists a TM transducer M with polynomial time complexity such that for any string s, $s \in B$ if and only if $M(s) \in A$. If a set A is PSPACE-complete then $A \in P$ if and only if $P = \mathrm{PSPACE}$. A well-known PSPACE-complete problem is the quantified Boolean formula (QBF) problem; namely, QBF = {all closed quantified Boolean formulas which are true}, under a fixed coding scheme for quantified Boolean formulas (Garey and Johnson, 1979; Stockmeyer and Meyer, 1973).

We use sequences of dyadic rational numbers to represent real numbers. A dyadic rational number is a rational number with a finite binary representation. Let D be the binary strings (with the binary point and signs) which represent dyadic rationals. We say a dyadic rational d is *of length* $\leqslant n$ if it has a binary representation with $\leqslant n$ digits to the right of the binary point. We say that a sequence $\{d_n\}$ of dyadic rationals *binary converges* to a real number x if for all n, $|d_n - x| \leqslant 2^{-n}$. A real number x is *computable* if there is a computable function ϕ: $\{0\}^* \to D$ such that $\{\phi(0^n)\}$ binary converges to x. x is *polynimial time* (*space*) *computable* if it is computable and the function ϕ above is polynomial time (space, respectively) computable. In other words, x is polynomial time computable if there is a TM M with polynomial time complexity such that for any input of the form 0^n, $M(0^n)$ outputs a dyadric rational d that approximates x close to within an error 2^{-n}.

A real function f: $[0, 1] \to R$ is *computable* if there is an oracle TM M such that for any oracle function ϕ: $\{0\}^* \to D$ which computes a sequence of dyadic rationals binary converging to a real number $x \in [0, 1]$, $\{M^{\phi}(0^n)\}$ is a sequence of dyadic rationals binary converging to $f(x)$. A real function f: $[0, 1] \to R$ is *polynomial time (space) computable* if it is computable by an oracle TM M with polynomial time (space, respectively) complexity. If f is polynomial time computable on $[0, 1]$, then f has a *polynomial modulus function* p such that for all $n \geqslant 0$, $|f(a) - f(b)| \leqslant 2^{-n}$ whenever $a, b \in [0, 1]$ and $|a - b| \leqslant 2^{-p(n)}$. The extension of the above definitions to functions on any domain of a compact interval or a 2-dimensional compact rectangle is straightforward.

The following results will be used in Section 4.

LEMMA 2.1. (a) $P_s = \mathrm{PSPACE}_s$ *iff* $P_{sf} = \mathrm{PSPACE}_{sf}$.

(b) $P_{sf} = \mathrm{PSPACE}_{sf}$ *implies that all polynomial space computable real numbers are polynomial time computable.*

Proof. The direction $P_{sf} = \mathrm{PSPACE}_{sf}$ implies $P_s = \mathrm{PSPACE}_s$ and (b) are obvious. We prove that $P_s = \mathrm{PSPACE}_s$ implies $P_{sf} = \mathrm{PSPACE}_{sf}$ in the following.

Let ϕ: $\{0\}^* \to \{0, 1\}^*$ be a function in PSPACE_{sf}. There is a polynomial α such that the length of $\phi(0^n) \leqslant \alpha(n)$ for all n.

Let tr: $\{0, 1\}^* \to \{0, 1\}^*$ be a simple translation function which maps each bit 0 to 01 and 1 to 10. Define ψ: $\{0\}^* \to \{0, 1\}^*$ by $\psi(0^n) = \mathrm{tr}(\phi(0^n)) \cdot 0^k$, where k is chosen such that length of $\psi(0^n)$ is exactly $2\alpha(n)$. Note that $\phi(0^n)$ can be decoded from $\psi(0^n)$ easily. Now define an infinite sequence $s = \psi(0)\,\psi(0^2)...$, and let A be the set $\subseteq \{0\}^*$ such that s is the characteristic sequence of A; i.e., $0^n \in A$ iff the nth bit in s is 1. We note that A is in PSPACE_s: to compute $\chi_A(0^n)$, we need only to compute $\phi(0),..., \phi(0^n)$ and construct an initial segment of s from them. So, A is in P_s. As a consequence, ψ is in P_{sf} (and so is ϕ): to compute $\psi(0^n)$, we need only to get $\chi_A(0^k)$ for all $k \leqslant \sum_{i=1}^{n} 2 \cdot \alpha(i)$. ∎

LEMMA 2.2. $P = \mathrm{PSPACE}$ *implies that all polynomial space computable real functions are polynomial time computable.*

Proof. Let g: $[0, 1] \to R$ be a polynomial space computable real function. Assume that g is computed by an oracle TM M with space complexity $\leqslant \alpha$, a polynomial. Then, M queries, for any oracle ϕ, at most $\phi(0),..., \phi(0^{\alpha(n)})$ because for each query $\phi(0^n)$, M needs at least n cells to copy the answer from ϕ. Now consider the following TM M_1. M_1 takes as input a pair $\langle s, 0^n \rangle$, $s \in \{0, 1\}^*$, and simulates the oracle TM M on input 0^n. Whenevery a query is made by M, M_1 simply uses s as the answer from the oracle. Then obviously M_1 has polynomial space complexity. From the assumption $P =$

PSPACE, there is an equivalent TM M_2 having polynomial time complexity. Let M_3 be the oracle TM which, on oracle ϕ and input 0^n, first queries ϕ to get $s = \phi(0^{\alpha(n)})$ then simulates M_2 on input $\langle s, 0^n \rangle$. Obviously, M_3 computes the function y and it has polynomial time complexity. So, y is polynomial time computable. ∎

Note that the converses of both lemmas are true. Indeed, they are corollaries of Theorems 3 and 4, (a) ⇒ (b). We also note that the converse of Lemma 2.1 actually has a short proof.

3. Two Negative Results

In this section we prove Theorems 1 and 2. Actually, Theorems 1 and 2 may be viewed as two corollaries of result (2). First we briefly review Pour-El and Richards's proof of result (2). The reader is referred to Pour-El and Richards (1979) for the details.

There are two major components in the construction of the function f. The first is a "fundamental box," which consists of the rectangle $[-c, c] \times [-c^2, c^2]$ and the function f_c on the rectangle for $c > 0$. The function f_c is made to satisfy the following properties:

(i) f is computable.

(ii) $\max\{|f(x, y)|: -c \leqslant x \leqslant c, -c^2 \leqslant y \leqslant c^2\} \leqslant 2c$, and $f(x, y) = 0$ on the boundary of the rectangle.

(iii) The equation $y'(x) = f_c(x, y)$ has a unique solution on $[-c, 0]$ but does not have a unique solution on $[0, c]$, provided the initial value $y(-c)$ is in $[-(\frac{99}{100})c^2, (\frac{99}{100})c^2]$.

(iv) The equation $y'(x) = f_c(x, y)$ on $[0, c] \times [-c^2, c^2]$ has a unique solution on $[0, c]$ if the initial value $y(0)$ is not equal to 0. Furthermore, $y(0) > 0$ implies $y(c) > (\frac{99}{100})c^2$ and $y(0) < 0$ implies $y(c) < -(\frac{99}{100})c^2$.

The second component is a "pulse" function $h_\alpha(x, y)$ with support $[-\alpha, \alpha] \times R$. $h_\alpha(x, y)$ is made to be computable and have the maximum value α at $x = 0$. $h_\alpha(x, y) \geqslant 0$ for all x, y.

Now the construction of the function f can be described as follows. Let A and B be a pair of recursively nonseparable sets of integers generated by one-to-one recursive functions a and b, respectively. That is, range$(a) = A$, range$(b) = B$, and there is no recursive set C such that $A \subseteq C$ and $B \subseteq \bar{C}$. Let $\alpha(n) = 2^{-(n+a(n)+5)}$ and $\beta(n) = 2^{-(n+b(n)+5)}$. Then define

$$f(x, z) = \sum_{m=0}^{\infty} f_{c_m}(x - v_m, z) + \sum_{n=0}^{\infty} h_{\alpha(n)}(x - v_{a(n)}, z)$$

$$- \sum_{n=0}^{\infty} h_{\beta(n)}(x - v_{b(n)}, z),$$

where $c_m = 2^{-(m+2)}$ and $v_m = 2^{-(m+1)} + 2^{-(m+2)}$. In other words, for each m, a fundamental box of size $2^{-(m+1)} \times 2^{-(2m+3)}$ is translated to the rectangle $[2^{-(m+1)}, 2^{-m}] \times [-c_m^2, c_m^2]$; further, a small positive pulse is added to it if $m \in A$, and a small negative pulse is added to it if $m \in B$. The pulses are so small that the maximum value of f on the mth box is $\leqslant 2c_m + 2^{-(m+5)} < 2^{-m}$. Therefore, f is computable. Moreover, from property (iv) of the function f_c, the values of any solution y of Eq. (1) at the right end of the boxes can be used to separate A from B, since a positive pulse in the box $[2^{-(m+1)}, 2^{-m}] \times [-c_m^2, c_m^2]$ makes $y(2^{-m}) > 0$ and a negative pulse in the box makes $y(2^{-m}) < 0$. Thus A and B would be recursively separable if any solution y is computable.

To proof Theorem 1, we simply observe that both f_c and h_α are polynomial time computable if c and α are. So, we modify the construction above as follows.

Let a and b be computed by Turing machines M_a and M_b in time ϕ and ψ, respectively. Replace $\alpha(n)$ by $\alpha'(n) = 2^{-(\phi(n)+a(n)+5)}$ and $\beta(n)$ by $\beta'(n) = 2^{-(\psi(n)+b(n)+5)}$. Define

$$f(x,z) = \sum_{m=0}^{\infty} f_{c_m}(x - v_m, z) + \sum_{n=0}^{\infty} h_{\alpha'(n)}(x - v_{a(n)}, z) - \sum_{n=0}^{\infty} h_{\beta'(n)}(x - v_{b(n)}, z).$$

Then, following the same argument, any solution y of Eq. (1) can still separate A from B. We only need to verify that f is polynomial time computable. Assume that we want to compute $f(x,z)$ correct to within an error 2^{-n}. First note that from the definition of the function f, given in Pour-El and Richards (1979), $f(x,z) = 0$ if $|x - 2^{-k}| \leqslant 2^{-(k+11)}$ for some $k > 0$. So, we first get an approximte value s of x such that $|s - x| \leqslant 2^{-(n+12)}$. If $s \leqslant 2^{-(n+1)}$ or $|s - 2^{-k}| \leqslant 2^{-(n+12)}$ for some $k > 0$, then output 0. Otherwise, find m such that $m \leqslant n+1$ and $2^{-(m+1)} < s < 2^{-m}$. Now simulate TM's M_a and M_b on input i, $0 \leqslant i \leqslant (n+1)$, for $n+1$ moves. If any one of them, say $M_a(i)$, halts in $n+1$ moves and outputs m, then compute $f(x,z)$ according to $f_{c_m}(x - v_m, z) + h_{\alpha'(i)}(x - v_m, z)$ (or, $f_{c_m}(x - v_m, z) - h_{\beta'(i)}(x - v_m, z)$, if $M_b(i) = m$); and if none of them halts in $n+1$ moves, then compute $f_{c_m}(x - v_m, z)$ (correct to within $2^{-(n+1)}$).

The correctness of the above algorithm follows from the observation that $a(i) = m$ and $\phi(i) > n+1$ implies $\alpha(i) < 2^{-(n+1)}$. The time complexity of the above algorithm is obviously bounded by a polynomial. So, Theorem 1 is proved.

To prove Theorem 2, we make another modification. For any recursive function ϕ let A be a recursive set such that for any TM M recognizing A,

the run time of $M(n)$ is greater than $2^{\phi(n)}$ for infinitely many n. Now let $\psi(n)$ be the run time of a TM M_A which recognizes A. Let $\alpha(n) = 2^{-(\psi(n)+n+5)}$. Define

$$f(x,z) = \sum_{m=0}^{\infty} f_{c_m}(x - v_m, z) + \sum_{m \in A} h_{\alpha(m)}(x - v_m, z) - \sum_{m \notin A} h_{\alpha(m)}(x - v_m, z).$$

That is, on the mth box, $f(x,z)$ is $f_{c_m}(x - v_m, z)$ plus a small pulse if $m \in A$, and minus a small pulse if $m \notin A$. First claim that f is still polynomial time computable. Note that the height of the pulse at the mth box is $\leqslant 2^{-(\psi(m)+5)}$. So, if we want to compute $f(x,z)$ correct to within an error 2^{-n}, with $x \in [2^{-(m+1)}, 2^{-m}]$, we can perform the computation of $M_A(m)$ for $n+1$ moves. If $M_A(m)$ halts in $n+1$ moves, we can compute, correct to within an error 2^{-n}, $f_{c_m}(x - v_m, z) \pm h_{\alpha(m)}(x - v_m, z)$. If $M_A(m)$ does not halt in $n+1$ moves, then $\alpha(m) < 2^{-(n+1)}$, and we can simply compute, correct to within an error $2^{-(n+1)}$, $f_{c_m}(x - v_m, z)$.

Now we observe that $m \in A$ implies $y(2^{-m}) > (\frac{1}{2}) \cdot c_m^2$ and $m \notin A$ implies $y(2^{-m}) < -(\frac{1}{2}) c_m^2$. So, whether $m \in A$ or not can be determined in time $\phi(2m+3) + O(m)$ if an approximate value of $y(x)$ within an error 2^{-n} can be computed in time $\phi(n)$. Thus y cannot be computed in time $\phi(n)$ in infinitely many places in $[0, 1]$.

Finally we remark that the following result follows from a simple modification of Miller's proof of result (3).

COROLLARY 2. *For any recursive real number a, $\frac{1}{4} \leqslant a \leqslant \frac{3}{4}$, there is a polynomial time computable real function f on $[0,1] \times [-1,1]$ such that $y(x) = ax^2$ is the unique solution of Eq.* (1).

4. Proof of Direction (b) $\Rightarrow$ (a) of Theorems 3 and 4

In this section we prove the direction (b) $\Rightarrow$ (a) of Theorems 3 and 4. We show that if a function f on $[0,1] \times [-1,1]$ satisfies the conditions of Theorem 3(a), then the solution y of Eq. (1) with respect to f is polynomial space computable. By Lemma 2.2, this proves Theorem 3, (b) $\Rightarrow$ (a). Furthermore, if y is a polynomial space computable function on $[0,1]$ then $y(1)$ is a polynomial space computable real number. So, Theorem 4, (b) $\Rightarrow$ (a), follows from Lemma 2.1.

We will show this result by a careful error analysis of Euler's method. First we define, for each $p > 0$, the Euler approximate function $\tilde{y}_p$ with increment 2^{-p}.

Let $\tilde{f}_p$ be a function defined on $\{(s,t)$: s and t are dyadic rationals, $s \in [0,1]$, and $t \in [-1,1]\}$ such that for any (s,t) in the domain, $\tilde{f}_p(s,t)$ is a dyadic rational of length $\leqslant p$ and $|\tilde{f}_p(s,t) - f(s,t)| \leqslant 2^{-p}$. (The existence of a polynomial time computable $\tilde{f}_p$ is guaranteed by the polynomial time computability of f.) $\tilde{y}_p$ is a piecewise linear function on $[0,1]$, with breakpoints in $\{k \cdot 2^{-p}$: $0 \leqslant k \leqslant 2^p\}$. For convenience, we write, in this section, $k^{(p)}$ to denote $k \cdot 2^{-p}$. At each point $k^{(p)}$, $\tilde{y}_p(k^{(p)})$ is a dyadic rational of length $\leqslant 2p$. We define

$$\tilde{y}_p(0) = 0,$$
$$\tilde{y}_p(x) = \tilde{y}_p(k^{(p)}) + \tilde{f}_p(k^{(p)}, \tilde{y}_p(k^{(p)})) \cdot (x - k^{(p)})$$

if $k^{(p)} < x \leqslant (k+1)^{(p)}$, $k = 0,\ldots, 2^p - 1$. The polynomial space computability of $\tilde{y}_p$ is obvious.

LEMMA 4.1. *Assume that f is polynomial time computable. Then, there is a polynomial function β_1 such that $\tilde{y}_p(k^{(p)})$ is computable in space $\beta_1(p)$.*

Next we need to show that for some polynomial function β_2, $p \geqslant \beta_2(n)$ implies $|\tilde{y}_p(x) - y(x)| \leqslant 2^{-n}$ for all $x \in [0,1]$. Recall that α is a polynomial function such that $\{(x,z)$: $0 \leqslant x \leqslant 1 - 2^{-n}$, $|z - y(x)| \leqslant 2^{-\alpha(n)}\} \subseteq E$. Let $M = \max\{|f(x,z)|$: $0 \leqslant x \leqslant 1$, $-1 \leqslant z \leqslant 1\} + 1$, $L \geqslant 1$ the Lipschitz constant for f, and γ a polynomial modulus function for f: for all $x_1, x_2 \in [0,1]$, $z_1, z_2 \in [-1,1]$ and any $n \geqslant 0$, $|f(x_1,z_1) - f(x_2,z_2)| \leqslant 2^{-n}$ whenever $|x_1 - x_2| \leqslant 2^{-\gamma(n)}$ and $|z_1 - z_2| \leqslant 2^{-\gamma(n)}$. Without loss of generality, assume that $\alpha(n) \geqslant n$ and $\gamma(n) \geqslant n$ for all $n \geqslant 0$.

Since $\tilde{f}_p$ and f are very close, M is also a bound for $|\tilde{f}_p(x,z)|$. From this bound M, we can immediately derive some upper bounds.

LEMMA 4.2. *Assume that $0 \leqslant x_1 < x_2 \leqslant 1$.*

(a) $|y(x_1) - y(x_2)| \leqslant M \cdot (x_2 - x_1)$.

(b) $|\tilde{y}_p(x_1) - \tilde{y}_p(x_2)| \leqslant M \cdot (x_2 - x_1)$.

(c) $|y(x_2) - \tilde{y}_p(x_2)| \leqslant |y(x_1) - \tilde{y}_p(x_1)| + 2M \cdot (x_2 - x_1)$.

These upper bounds will be used in our proof. From Lemma 4.2(c), for any n and p, if $|\tilde{y}_p(x_n) - y(x_n)| \leqslant 2^{-(n+1)}$ at $x_n = 1 - 2^{-(n+2+\lceil \log_2 M \rceil)}$ then $|\tilde{y}_p(x) - y(x)| \leqslant 2 \cdot M \cdot 2^{-(n+2+\lceil \log_2 M \rceil)} + 2^{-(n+1)} \leqslant 2^{-n}$ for all $x \in [x_n, 1]$. Therefore, we need only to show that for some polynomial function β_2, $p \geqslant \beta_2(n)$ implies $|\tilde{y}_p(x) - y(x)| \leqslant 2^{-(n+1)}$ for all $x \in [0, x_n]$.

In the following we fix n and consider $x \in [0, x_n]$. Let $j = \alpha(n + 2 + \lceil \log_2 M \rceil)$ and $\beta_3(m) = \gamma(m+2) + \lceil \log_2 M \rceil + 2$. We claim that for all

$x \in [0, x_n]$, $|\tilde{y}_p(x) - y(x)| \leqslant 2^{-m} \cdot (e^{Lx} - 1) \cdot L^{-1} \leqslant 2^{-(m-2L)}$ if $p \geqslant \beta_3(m)$ and $m \geqslant j + 2L + 1$. In other words, $\beta_2(n) = \beta_3(j + 2L + 1)$ suffices.

Let $p \geqslant \beta_3(m)$ and $d(x) = y(x) - \tilde{y}_p(x)$. We show the above claim by induction. That is, we show that for each integer k such that $k^{(p)} < x_n$, $|d(x)| \leqslant 2^{-m} \cdot (e^{Lx} - 1) \cdot L^{-1}$ for all $x \in (k^{(p)}, (k+1)^{(p)}] \cap [0, x_n]$, assuming that $|d(k^{(p)})| \leqslant 2^{-m} \cdot (e^{Lk^{(p)}} - 1) \cdot L^{-1}$.

We consider the following three cases (Lemmas 4.3–4.5).

LEMMA 4.3. *If* $|d(k^{(p)})| \leqslant M \cdot 2^{-(p-1)}$, *then* $|d(x)| \leqslant |d(k^{(p)})| + 2^{-(m+1)} \cdot (x - k^{(p)})$, *for all* $x \in (k^{(p)}, (k+1)^{(p)}]$.

Proof. We note that

$$|d(x)| \leqslant |d(k^{(p)})| + \int_{k^{(p)}}^{x} |f(t, y(t)) - \tilde{f}_p(k^{(p)}, \tilde{y}_p(k^{(p)}))|\, dt$$

$$\leqslant |d(k^{(p)})| + \int_{k^{(p)}}^{x} |f(t, y(t)) - f(k^{(p)}, \tilde{y}_p(k^{(p)}))|\, dt$$

$$+ \int_{k^{(p)}}^{x} |f(k^{(p)}, \tilde{y}_p(k^{(p)})) - \tilde{f}_p(k^{(p)}, \tilde{y}_p(k^{(p)}))|\, dt.$$

The second term of the last expression is $\leqslant 2^{-(m+2)} \cdot (x - k^{(p)})$ because $t \in [k^{(p)}, (k+1)^{(p)}]$ implies $[|t - k^{(p)}| \leqslant 2^{-p} \leqslant 2^{-\gamma(m+2)}$ and $|y(t) - \tilde{y}_p(k^{(p)})| \leqslant |y(t) - y(k^{(p)})| + |d(k^{(p)})| \leqslant M \cdot 2^{-p} + M \cdot 2^{-(p-1)} \leqslant 2^{-\gamma(m+2)}]$. Also, from the definition of $\tilde{f}_p$, the third term is $\leqslant 2^{-(m+2)} \cdot (x - k^{(p)})$. Putting them together, we have proved Lemma 4.3. ∎

LEMMA 4.4. *If* $d(k^{(p)}) > M \cdot 2^{-(p-1)}$ *then* $0 < d(x) \leqslant d(k^{(p)}) \cdot e^{L(x-k^{(p)})} + 2^{-m} \cdot (e^{L(x-k^{(p)})} - 1) \cdot L^{-1}$.

Proof. First we check that

$$d(x) = d(k^{(p)}) + \int_{k^{(p)}}^{x} (f(t, y(t)) - \tilde{f}_p(k^{(p)}, \tilde{y}_p(k^{(p)})))\, dt$$

$$\geqslant d(k^{(p)}) - 2 \cdot M \cdot (x - k^{(p)})$$

$$\geqslant d(k^{(p)}) - M \cdot 2^{-(p-1)}$$

$$> 0.$$

Next, we reduce $d(x)$ as follows.

$$d(x) = d(k^{(p)}) + \int_{k^{(p)}}^{x} (f(t, y(t)) - \tilde{f}_p(k^{(p)}, \tilde{y}_p(k^{(p)})))\, dt$$

$$= d(k^{(p)}) + \mathrm{I} + \mathrm{II} + \mathrm{III},$$

where

$$\mathrm{I} = \int_{k^{(p)}}^{x} (f(t, y(t)) - f(t, \tilde{y}_p(t)))\, dt,$$

$$\mathrm{II} = \int_{k^{(p)}}^{x} (f(t, \tilde{y}_p(t)) - f(k^{(p)}, \tilde{y}_p(k^{(p)})))\, dt,$$

and

$$\mathrm{III} = \int_{k^{(p)}}^{x} (f(k^{(p)}, \tilde{y}_p(k^{(p)})) - \tilde{f}_p(k^{(p)}, \tilde{y}_p(k^{(p)})))\, dt.$$

We will use the right Lipschitz condition to control I, the modulus function γ to control II, and the error bound of $\tilde{f}_p$ to control III.

From the inductive hypothesis that $|d(k^{(p)})| \leqslant 2^{-(m-2L)}$, we have, for $x \in [k^{(p)}, (k+1)^{(p)}]$, $|d(x)| \leqslant |d(k^{(p)})| + 2 \cdot M \cdot (x - k^{(p)}) \leqslant 2^{-(m-2L)} + 2^{-\gamma(m+2)} \leqslant 2^{-(m-2L-1)} \leqslant 2^{-j}$. Thus, for any $x \in [k^{(p)}, (k+1)^{(p)}]$, the line segment connecting $(x, y(x))$ and $(x, \tilde{y}_p(x))$ is contained in E because $j = \alpha(n + 2 + \lceil \log_2 M \rceil)$. Also, we have just proved, in the first half of this lemma, that $y(x) > \tilde{y}_p(x)$ for all $x \in [k^{(p)}, (k+1)^{(p)}]$. Therefore, by the right Lipschitz condition of f, we have $\mathrm{I} \leqslant L \cdot \int_{k^{(p)}}^{x} d(t)\, dt$.

Next we check that for all $t \in [k^{(p)}, (k+1)^{(p)}]$, $|t - k^{(p)}| \leqslant 2^{-p} \leqslant 2^{-\gamma(m+2)}$, and $|\tilde{y}_p(t) - \tilde{y}_p(k^{(p)})| \leqslant M \cdot 2^{-p} \leqslant 2^{-\gamma(m+2)}$. This implies that $\mathrm{II} \leqslant 2^{-(m+2)} \cdot (x - k^{(p)})$.

Finally, from the definition of $\tilde{f}_p$, we have $\mathrm{III} \leqslant 2^{-p} \cdot (x - k^{(p)}) \leqslant 2^{-m} \cdot (x - k^{(p)})$.

Together, we get

$$d(x) \leqslant d(k^{(p)}) + L \cdot \int_{k^{(p)}}^{x} d(t)\, dt + 2^{-m}(x - k^{(p)}). \tag{2}$$

Define the function $g(x) = \int_{k^{(p)}}^{x} d(t)\, dt$ on $[k^{(p)}, (k+1)^{(p)}]$ and let $\Delta x = x - k^{(p)}$. Rewriting the above inequality (2) we have

$$g'(x) - L \cdot g(x) \leqslant d(k^{(p)}) + 2^{-m} \cdot \Delta x.$$

Multiplying both sides by $e^{-L \cdot \Delta x}$ and integrating the resulting expression from $k^{(p)}$ to x, we obtain

$$e^{-L \cdot \Delta x} \cdot g(x) \leqslant d(k^{(p)}) \cdot (1 - e^{-L \cdot \Delta x}) \cdot L^{-1} - 2^{-m} \cdot e^{-L \cdot \Delta x} \cdot (1 + L \cdot \Delta x) \cdot L^{-2} + 2^{-m} \cdot L^{-2},$$

or

$$g(x) \leqslant d(k^{(p)}) \cdot (e^{L \cdot \Delta x} - 1) \cdot L^{-1} - 2^{-m}(1 + L \cdot \Delta x) \cdot L^{-2} + 2^{-m} \cdot e^{L \cdot \Delta x} \cdot L^{-2}.$$

Combining this with (2), the simplified inequality is

$$d(x) \leqslant d(k^{(p)}) \cdot e^{L \cdot \Delta x} + 2^{-m} \cdot (e^{L \cdot \Delta x} - 1) \cdot L^{-1}$$

which is the desired result. ∎

LEMMA 4.5. *If* $d(k^{(p)}) < -M \cdot 2^{-(p-1)}$, *then* $d(k^{(p)}) \cdot e^{L \cdot (x - k^{(p)})} - 2^{-m} \cdot (e^{L \cdot (x - k^{(p)})} - 1) \cdot L^{-1} \leqslant d(x) < 0$, *for all* $x \in [k^{(p)}, (k+1)^{(p)}]$.

Proof. Symmetric to the proof of Lemma 4.4. ∎

So, for the first case, if $|d(k^{(p)})| \leqslant M \cdot 2^{-(p-1)}$, we have, from Lemma 4.3, $|d(x)| \leqslant |d(k^{(p)})| + 2^{-(m+1)} \cdot (x - k^{(p)})$ for $x \in [k^{(p)}, (k+1)^{(p)}]$. From the inductive hypothesis, $|d(x)| \leqslant 2^{-m} \cdot (e^{L \cdot k^{(p)}} - 1) \cdot L^{-1} + 2^{-m} \cdot ((x - k^{(p)})/2) \leqslant 2^{-m} \cdot (e^{L \cdot k^{(p)}} - 1) \cdot L^{-1} + 2^{-m} \cdot (e^{L \cdot x} - e^{L \cdot k^{(p)}}) \cdot L^{-1} = 2^{-m} \cdot (e^{Lx} - 1) \cdot L^{-1}$, because $e^a - e^b \geqslant (a-b)/2$ for any a, b such that $a > b \geqslant 0$.

For the second case, if $|d(k^{(p)})| > M \cdot 2^{-(p-1)}$, we have, from Lemmas 4.4 and 4.5, $|d(x)| \leqslant |d(k^{(p)})| \cdot e^{L(x - k^{(p)})} + 2^{-m} \cdot (e^{L(x-k^{(p)})} - 1) \cdot L^{-1} \leqslant 2^{-m} \cdot (e^{Lk^{(p)}} - 1) \cdot e^{L(x-k^{(p)})} \cdot L^{-1} + 2^{-m} \cdot (e^{L(x-k^{(p)})} - 1) \cdot L^{-1} = 2^{-m} \cdot (e^{Lx} - 1) \cdot L^{-1}$.

This completes our inductive step of the proof. Since $d(0^{(p)}) = 0$, the induction proof is also done, and so is the proof of (b) ⇒ (a) of Theorem 3.

5. THE MAIN RESULTS

In this section we prove the direction (a) ⇒ (b) of Theorems 3 and 4. The proofs of the two theorems share a common basic construction of a real function, based on the idea of the alpha–beta pruning algorithm (Knuth and Moore, 1975) for the self-reducing tree of a quantified Boolean formula, (Meyer and Paterson, 1979; Ko, 1983). (Strictly speaking, the algorithm we will describe is simpler than the real alpha–beta pruning and is called by Knuth and Moore (1975) as the branch and bound algorithm.) In the following we first give a sketch of the proofs in Section 5.1. Then, in Section 5.2 we give the basic construction, which will be used in Sections 5.3 and 5.4 for the proof of (a)) ⇒ (b) of Theorems 3 and 4, respectively.

5.1. *Sketch of the Proofs*

Since the QBF problem is known as PSPACE-complete, our goal is to construct a real function f of $[0, 1] \times [-1, 1]$ such that the solution y of Eq. (1) with respect to f is unique and it encodes the information about the QBF problem. In Section 5.2, we describe how to construct, for each quantified Boolean formula G with p quantifiers, a real function f_G on $[0, 1] \times [-1, 1]$ which satisfies the following properties:

(i) f_G is polynomial time computable;

(ii) there is a set $E \subseteq [0, 1] \times [-1, 1]$, and a polynomial function α, such that (a) f_G satisfies the right Lipschitz condition on E, (b) the solution y of Eq. (1) with respect to f_G is unique, and (c) $\{(x, z)\colon 0 \leqslant x \leqslant 1, |z - y(x)| \leqslant 2^{-\alpha(p)}\} \subseteq E$; and

(iii) $y(1) > 0$ if and only if G is TRUE.

With this basic construction, we can construct a more complicated function f to satisfy the conditions given in Theorem 3(a) as follows. We divide the unit interval $[0, 1]$ into infinitely many subintervals, each representing one formula. Let J_G represent the formula G. The rectangle $J_G \times [-1, 1]$ is further divided horizontally into many subrectangles. On each subrectangle $J_G \times [c, d]$, f is defined to behave similarly to f_G on $[0, 1] \times [-1, 1]$ such that the solution y of Eq. (1) with respect to f on J_G has the property that the value of y at the right endpoint of J_G is $> (c + d)/2$ if and only if G is TRUE. By a careful arrangement of the sizes of the subrectangles, we can show that the truth value of G can be computed easily from the value of y at the right endpoint of J_G. Thus the polynomial time computability of y implies QBF $\in P$, and hence $P =$ PSPACE.

For Theorem 4, a similar construction is to be made. For each set $A \in \text{PSPACE}_s$, we let ϕ be a reduction function for $A \leqslant_m^p$ QBF. That is, for each 0^k, $\phi(0^k)$ is a quantified Boolean formula G_k such that $0^k \in A$ if and only if G_k is TRUE. We divide $[0, 1]$ into infinitely many subintervals, each representing a formula G_k. Similar to the proof of Theorem 3, we divide the rectangle $J_k \times [-1, 1]$, where J_k is the interval representing G_k, into subrectangles. We define f on each subrectangle $J_k \times [c, d]$ to behave like f_{G_k}. Also, we arrange the sizes of subrectangles in such a way that if a solution y of Eq. (1) with respect to f leaves a subrectangle $J_k \times [c, d]$ on the upper half, then $(c + d)/2 < y(1) < d$. Thus, by decoding $y(1)$, we can determine whether G_k is TRUE for all G_k. That is, the set $A \in P_s$ if $y(1)$ is polynomial time computable.

5.2. *The Basic Construction*

Let G be an arbitrary, but fixed quantified Boolean formula. We want to construct a function $f = f_G$ satisfying the conditions (i)–(iii) given in Section 5.1. To make the idea clear, we first describe a "discrete version" of an initial value problem based on the formula G. Then a real function f can be defined from this discrete version of the initial value problem.

We write G in its normal form:

$$G = (Q_1 t_1) \cdots (Q_p t_p)\, T(t_1, \ldots, t_p)$$

where $p \geqslant 0$, $Q_1,..., Q_p \in \{\exists, \forall\}$ and T is a quantifier-free, polynomial time computable predicate with variables $t_1,..., t_p$. Then the self-reducing tree of G can be described as follows.

(i) The root of the tree is G.

(ii) Each nonterminal node of the tree has the form $G(s_1 \cdots s_m) = (Q_{m+1} t_{m+1}) \cdots (Q_p t_p)\ T(s_1,..., s_m, t_{m+1},..., t_p)$ with $m < p$ and $s_1,..., s_m \in \{0, 1\}$. A nonterminal node $G(s_1 \cdots s_m)$ has two sons: $G(s_1 \cdots s_m 0)$ and $G(s_1 \cdots s_m 1)$. It is an existential node if $Q_{m+1} = \exists$, and a universal node if $Q_{m+1} = \forall$.

(iii) A node which contains a quantifier-free formula $G(s_1 \cdots s_p)$ is a terminal node.

To determine the truth value of a formula G, we consider the following polynomial space-bounded algorithm for QBF, which is usually called the alpha–beta pruning algorithm.

The algorithm performs a depth-first search over the self-reducing tree of G. At each node $G(s_1 \cdots s_m)$, we first search its left subtree. If $G(s_1 \cdots s_m 0)$ is TRUE and $Q_{m+1} = \forall$, or if $G(s_1 \cdots s_m 0)$ is FALSE and $Q_{m+1} = \exists$, then we continue to search over its right son, and $G(s_1 \cdots s_m)$ is TRUE if and only if $G(s_1 \cdots s_m 1)$ is TRUE. On the other hand, if $G(s_1 \cdots s_m 0)$ is TRUE and $Q_{m+1} = \exists$, or if $G(s_1 \cdots s_m 0)$ is FALSE and $Q_{m+1} = \forall$, then $G(s_1 \cdots s_m)$ is TRUE if and only if $G(s_1 \cdots s_m 0)$ is TRUE, and we can skip the right subtree of $G(s_1 \cdots s_m)$.

The above is the most common way to describe the alpha–beta pruning algorithm using recursive calls. To fit our construction, we need to formulate it differently. We may view the algorithm as consisting of 2^p stages. At each stage, we evaluate the truth value of one terminal node and perform the pruning. It is important in our construction to know when *not* to evaluate a terminal node (because it has been pruned in some previous stage). To see whether a node has been deleted, we need only to know which node is the "rightmost" node whose truth value is already known. If this node is an ancestor of the terminal node then the terminal node has been deleted. This observation leads to the following pidgin-PASCAL algorithm, which does not make recursive calls. Let *pred* and *succ* be the predecessor and successor functions on $\{0, 1\}^*$, respectively, defined by the natural lexicographic order. In the algorithm, $v_G(s)$ will denote the lowest ancestor of the node $G(s)$ whose truth value can be determined by that of $G(0^p),..., G(s)$; u_i will denote the truth value of $G(s_1 \cdots s_i)$ (for a fixed string $s = s_1 \cdots s_p$).

for $s := 0^p$ **to** 1^p **do**
begin
  if $s = 0^p$ **or** $v_G(\text{pred}(s))$ is not an ancestor of $G(s)$
  then begin

```
        u_p := truth value of G(s);
        {write s = s_1 ··· s_p with s_1,..., s_p ∈ {0, 1}.}
        for i := p downto 0 do
        begin
            delete G(s_1 ··· s_i);
            if i = 0 then goto 2;
        1:  if s_i = 1 then u_{i-1} := u_i
            else if (u_i = TRUE and Q_i = ∃)
                 or (u_i = FALSE and Q_i = ∀)
                 then u_{i-1} := u_i
            else goto 2
        end;
    2:  v_G(s) := G(s_1 ··· s_i)
    end
else v_G(s) := v_G(pred(s))
end.
```

Remarks. (a) At label 1, $s_i = 1$ means that $G(s_1 \cdots s_i)$ is the right son of $G(s_1 \cdots s_{i-1})$. That means $G(s_1 \cdots s_{i-1}0)$ must have been deleted and the truth value of $G(s_1 \cdots s_{i-1}0)$ cannot determine that of $G(s_1 \cdots s_{i-1})$. Therefore, $G(s_1 \cdots s_{i-1})$ is TRUE if and only if $G(s_1 \cdots s_i)$ is TRUE.

(b) Note that this algorithm is not as efficient as the original alpha–beta pruning algorithm as this algorithm always uses an exponential amount of time to determine the truth value of G whereas the original algorithm may terminate quickly in some cases. However, our goal here is not to improve the alpha–beta pruning algorithm, but to formulate it into a "discrete version" of the initial value problem.

In order to make the application of this algorithm to the construction of the function f easy, we need to further modify it. (The reasons for these modifications will become clear later.)

(a) In the above algorithm, we deleted, at each stage, as many nodes as we could. This is actually not necessary, as we need only to delete "enough" nodes so that the question of whether the next terminal node needs to be evaluated can be determined from $v_G(s)$.

(b) We need to use an integer representation for the node $v_G(s)$ which satisfies the following condition: if G and H are two formulas with the same quantifiers, and having the same truth value at s, then $v_G(\text{pred}(s)) < v_H(\text{pred}(s))$ implies $v_G(s) < v_H(s)$.

(c) We need to have a polynomial time algorithm to compute $v_G(s)$ from $v_G(\text{pred}(s))$, the truth value of $G(s)$ and the formula G.

We describe, in the following, the definition of an integer function v_G

which satisfies our requirements. Let $\{0, 1\}^p$ be the set of all binary strings of length p. Let A_p be the set $\{n \in Z\colon -(p+1) \leqslant n \leqslant p+1,\ n \neq 0\}$. v_G is a function from $\{0, 1\}^p$ to A_p. It is defined recursively.

First, if G has no variable ($p = 0$), then define

$$\begin{aligned} v_G(\varepsilon) &= 1 && \text{if } G \text{ is TRUE,} \\ &= -1 && \text{if } G \text{ is FALSE.} \end{aligned}$$

Assume that G has $p > 0$ quantifiers starting with $Q_1 = \exists$. Then, for each string $s_2 \cdots s_p \in \{0, 1\}^{p-1}$, define

$$v_G(0s_2 \cdots s_p) = v_{G(0)}(s_2 \cdots s_p),$$

and

$$\begin{aligned} v_G(1s_2 \cdots s_p) &= p+1 && \text{if } G(0) \text{ is TRUE,} \\ &= v_{G(1)}(s_2 \cdots s_p) && \text{if } G(0) \text{ is FALSE.} \end{aligned}$$

Assume that G has $p > 0$ quantifiers starting with $Q_1 = \forall$. Then, for each $s_2 \cdots s_p \in \{0, 1\}^{p-1}$, define

$$v_G(0s_2 \cdots s_p) = v_{G(0)}(s_2 \cdots s_p),$$

and

$$\begin{aligned} v_G(1s_2 \cdots s_p) &= v_{G(1)}(s_2 \cdots s_p) && \text{if } G(0) \text{ is TRUE,} \\ &= -(p+1) && \text{if } G(0) \text{ is FALSE.} \end{aligned}$$

EXAMPLE 5.1. Assume that $G = (\forall t_1)\ (\forall t_2)\ (\exists t_3)\ T(t_1, t_2, t_3)$, with $T(0, 0, 0) = T(0, 1, 0) =$ TRUE and $T(0, 0, 1) =$ FALSE. Then we have

$$\begin{aligned} v_G(000) &= v_{G(0)}(00) = v_{G(00)}(0) = v_{G(000)}(\varepsilon) \\ &= 1, \text{ because } G(000) \text{ is TRUE;} \\ v_G(001) &= v_{G(0)}(01) = v_{G(00)}(1) \\ &= 2, \text{ because } G(00) \text{ starts with } \exists \text{ and } G(000) = \text{TRUE;} \\ v_G(010) &= v_{G(0)}(10) \\ &= v_{G(01)}(0), \text{ because } G(0) \text{ starts with } \forall \text{ and } G(00) \text{ is TRUE,} \\ &= v_{G(010)}(\varepsilon) = 1; \end{aligned}$$

and

$$\begin{aligned} v_G(011) &= v_{G(0)}(11) = v_{G(01)}(1) \\ &= 2, \text{ because } G(01) \text{ starts with } \exists \text{ and } G(010) \text{ is TRUE.} \end{aligned}$$

Intuitively, the value of $v_G(s_1 \cdots s_p)$ contains two pieces of information: the level of a node (the distance between a node and leaves) in the tree which can be used to determine whether the next terminal node $G(\text{succ}(s_1 \cdots s_p))$ needs to be evaluated or not (by the magnitude of the value), and the truth value of that node (by the sign of the value). We formulate this intuitive notion in the following lemmas. (The proofs of Lemmas 5.1–5.3 are included in the Appendix.)

LEMMA 5.1. *Let G be a formula with $p > 0$ quantifiers, and $s_1 \cdots s_p \in \{0, 1\}^p - \{0^p\}$. Also let $k = p + 1 - |v_G(s_1 \cdots s_p)|$. Then $v_G(s_1 \cdots s_p) > 0$ if and only if $G(s_1 \cdots s_k)$ is* TRUE.

In particular, we have

LEMMA 5.2. *Let G be a formula with p quantifiers. Then $v_G(1^p) > 0$ if and only if G is* TRUE.

Lemmas 5.1 and 5.2 showed that our definition of v_G can be used in the modified alpha–beta pruning algorithm. For example, the condition

$$v_G(\text{pred}(s)) \text{ is not an ancestor of } G(s)$$

in the third line of the modified alpha–beta pruning algorithm may be replaced by

$$t := \text{pred}(s); \qquad k := p + 1 - |v_G(t)|;$$
$$G(t_1 \cdots t_k) \text{ is not an ancestor of } G(s).$$

In addition to this, we also need to know how to compute $v_G(s)$ from $v_G(\text{pred}(s))$. The following lemma shows the existence of a simple algorithm for this.

LEMMA 5.3. *Let G have $p > 0$ quantifiers. For any $s_1 \cdots s_p \in \{0, 1\}^p - \{0^p\}$, let $m = m(s_1 \cdots s_p) = \max\{i: 1 \leqslant i \leqslant p \text{ and } s_i = 1\}$. Let Q_m be the mth quantifier of G from the left.*

(a) $|v_G(\text{pred}(s_1 \cdots s_p))| > p + 1 - m$ *implies* $v_G(s_1 \cdots s_p) = v_G(\text{pred}(s_1 \cdots s_p))$.

(b) *If* $Q_m = \exists$ *then* $0 < v_G(\text{pred}(s_1 \cdots s_p)) \leqslant p + 1 - m$ *implies* $v_G(s_1 \cdots s_p) = p + 2 - m$, *and* $-(p + 1 - m) \leqslant v_G(\text{pred}(s_1 \cdots s_p)) < 0$ *implies*

$$v_G(s_1 \cdots s_p) = 1 \qquad \text{if } G(s_1 \cdots s_p) \text{ is TRUE,}$$
$$= -1 \qquad \text{if } G(s_1 \cdots s_p) \text{ is FALSE.}$$

(c) *If* $Q_m = \forall$ *then* $0 < v_G(\text{pred}(s_1 \cdots s_p)) \leqslant p + 1 - m$ *implies*

$$v_G(s_1 \cdots s_p) = 1 \qquad \text{if } G(s_1 \cdots s_p) \text{ is TRUE,}$$
$$= -1 \qquad \text{if } G(s_1 \cdots s_p) \text{ is FALSE,}$$

and $-(p + 1 - m) \leqslant v_G(\text{pred}(s_1 \cdots s_p)) < 0$ *implies* $v_G(s_1 \cdots s_p) = -(p + 2 - m)$.

The reader should compare the above relation between $v_G(\text{pred}(s))$ and $v_G(s)$ with the one defined in the modified alpha–beta pruning algorithm.

From Lemma 5.3, we define the function w_G: $\{0, 1\}^p \times A_p \to A_p$ by the following pidgin-PASCAL algorithm.

(Algorithm for w_G).

input: $(s_1 \cdots s_p, k)$
Begin
  if $s_1 = \cdots = s_p = 0$ **then output** k
  else begin
    $m := \max\{i: 1 \leqslant i \leqslant p, s_i = 1\}$;
    case
      $|k| > p + 1 - m$: **output** k;
      $0 < k \leqslant p + 1 - m$ **and** $Q_m = \exists$: **output** $p + 2 - m$;
      $-(p + 1 - m) \leqslant k < 0$ **and** $Q_m = \forall$: **output** $-(p + 2 - m)$;
      else: **if** $G(s_1 \cdots s_p)$ is TRUE **then output** 1
        else output -1;
    end {case}
  end
End.

From Lemma 5.3, for any $s_1 \cdots s_p \in \{0, 1\}^p - \{0^p\}$, $w_G(s_1 \cdots s_p, v_G(\text{pred}(s_1 \cdots s_p))) = v_G(s_1 \cdots s_p)$. Also it is obvious that there is a polynomial β such that $w_G(s_1 \cdots s_p, k)$ is computable in $\beta(p)$ steps.

Before we construct the function f, we remark that the original alpha–beta algorithm now can be modified as follows:

for $s := 0^p$ **to** 1^p **do**
begin
  if $s = 0^p$ **then if** $G(0_p) = \text{TRUE}$
    then $v_G(s) := 1$ **else** $v_G(s) := -1$
  else $v_G(s) := w_G(s, v_G(\text{pred}(s)))$
end.

Construction of f. The function f constructed here, intuitively, simulates the discrete function w_G such that the solution y of Eq. (1) with respect to f has the property that $y(0.s_1 \cdots s_p + 2^{-p})$ encodes the value $v_G(s_1 \cdots s_p)$.

Since $v_G(1^p)$ determines the truth value of G, we will have $y(1) > 0$ if and only if G is TRUE.

First we define two basic components of the function f. We call them an upward box and a downward box. An upward box consists of a rectangle $R = [x_1, x_2] \times [z_1, z_2]$, with two parameters h_1 and h_2 satisfying $2h_1 + h_2 \leqslant z_2 - z_1$, and a function g defined on R. The rectangle is divided into three areas: $R_1 = [x_1, x_2] \times [z_1, z_1 + h_1]$, $R_2 = [x_1, x_2] \times [z_1 + h_1, z_2 - h_1]$, and $R_3 = [x_1, x_2] \times [z_2 - h_1, z_2]$. The function g on R_2 is defined as follows:

$$\begin{aligned} g(x, z) &= a(x - x_1) \quad \text{if} \quad x_1 \leqslant x \leqslant (x_1 + x_2)/2, \\ &= a(x_2 - x) \quad \text{if} \quad (x_1 + x_2)/2 < x \leqslant x_2, \end{aligned}$$

where $a = 4(z_2 - z_1 - 2h_1 - h_2)/(x_2 - x_1)^2$. The areas R_1 and R_3 are interpolation areas:

$$\begin{aligned} g(x, z) &= g(x, z_2 - h_1) \cdot (z_2 - z) \cdot h_1^{-1} \quad \text{if} \quad z_2 - h_1 < z \leqslant z_2, \\ &= g(x, z_1 + h_1) \cdot (z - z_1) \cdot h_1^{-1} \quad \text{if} \quad z_1 \leqslant z < z_1 + h_1. \end{aligned}$$

In an upward box, the solution y of the equation $y'(x) = g(x, y(x))$ has a unique solution on $[x_1, x_2]$. Also, the relations between $y(x_1)$ and $y(x_2)$ are as follows:

$$y(x_1) \in [z_1, z_1 + h_1] \quad \text{implies} \quad y(x_2) \in [z_1, z_2 - h_2 - h_1]$$

$$y(x_1) \in [z_1 + h_1, z_1 + h_1 + h_2] \quad \text{implies} \quad y(x_2) \in [z_2 - h_2 - h_1, z_2 - h_1]$$

and

$$y(x_1) \in [z_1 + h_1 + h_2, z_2] \quad \text{implies} \quad y(x_2) \in [z_2 - h_1, z_2].$$

The relations are shown in Fig. 1a.

The downward box is similarly defined. To be more precise, let $g(x, z)$ on the rectangle R be an upward box with parameters h_1 and h_2. Then a

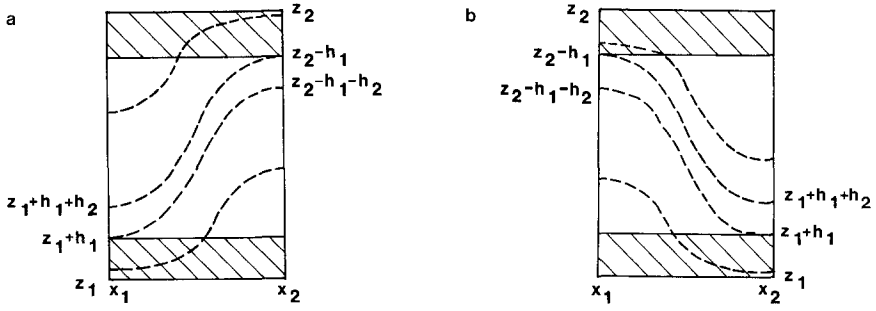

FIG. 1. An upward box (a) and a downward box (b). The value $g(x, z)$ is independent of z on the middle portion (unshaded areas). The shaded areas are interpolation areas with $g(x, z) = 0$ on the boundary. The dashed curves are the solutions of the equation.

downward box on the rectangle R with parameters h_1 and h_2 can be simply defined by the function $g_1(x, z) = -g(x, z)$. The relations between $y(x_1)$ and $y(x_2)$ of the solution y of the equation $y'(x) = g_1(x, y(x))$ on a downward box are shown in Fig. 1b.

Now we are ready for the definition of $f = f_G$. Assume that the formula G has $p > 0$ many quantifiers. (For the formulas G with no quantifier, the function f_G is trivial and we omit this case.) We divide the unit interval $[0, 1]$ into 2^p many subintervals of equal length. We call a subinterval an $(s_1 \cdots s_p)$-interval, with $s_1 \cdots s_p \in \{0, 1\}^p$, if the binary representation of the left endpoint of the subinterval is $0.s_1 \cdots s_p$, and call the rectangle $[0.s_1 \cdots s_p,\ 0.s_1 \cdots s_p + 2^{-p}] \times [-1, 1]$ the $(s_1 \cdots s_p)$-rectangle. Our goal is to define the function f on the $(s_1 \cdots s_p)$-rectangle such that the relations between $y(0.s_1 \cdots s_p)$ and $y(0.s_1 \cdots s_p + 2^{-p})$ of the solution y of Eq. (1) are similar to the relations between integers $k \in A_p$ and $w_G(s_1 \cdots s_p, k)$.

Note that for each $s_1 \cdots s_p \in \{0, 1\}^p - \{0^p\}$, there are only four types of w_G functions. Let $m = m(s_1 \cdots s_p) = \max\{i:\ 1 \leqslant i \leqslant p,\ s_i = 1\}$. Then, depending upon the mth quantifier Q_m of G and the truth value of $G(s_1 \cdots s_p)$, the function $\lambda k[w_G(s_1 \cdots s_p, k)]$ may be of one of the following four types:

Type 1.

$$\begin{aligned} w_G(s_1 \cdots s_p, k) &= k && \text{if } p + 2 - m \leqslant |k|, \\ &= p + 2 - m && \text{if } 0 < k < p + 2 - m, \\ &= 1 && \text{if } -(p + 2 - m) < k < 0. \end{aligned}$$

Type 2.

$$\begin{aligned} w_G(s_1 \cdots s_p, k) &= k && \text{if } p + 2 - m \leqslant |k|, \\ &= p + 2 - m && \text{if } 0 < k < p + 2 - m, \\ &= -1 && \text{if } -(p + 2 - m) < k < 0. \end{aligned}$$

Type 3.

$$\begin{aligned} w_G(s_1 \cdots s_p, k) &= k && \text{if } p + 2 - m \leqslant |k|, \\ &= 1 && \text{if } 0 < k < p + 2 - m, \\ &= -(p + 2 - m) && \text{if } -(p + 2 - m) < k < 0. \end{aligned}$$

Type 4.

$$\begin{aligned} w_G(s_1 \cdots s_p, k) &= k && \text{if } p + 2 - m \leqslant |k|, \\ &= -1 && \text{if } 0 < k < p + 2 - m, \\ &= -(p + 2 - m) && \text{if } -(p + 2 - m) < k < 0. \end{aligned}$$

For each type of w_G, we describe the corresponding f on the $(s_1 \cdots s_p)$-rectangle. We fix a parameter $\delta_p = (8n)^{-1}$ for some $n > p+2$. We divide $[-1, 1]$ into $16 \cdot \delta_p^{-1}$ many subintervals.

We define a function I_p: $A_p \to \{\text{subintervals of } [-1, 1]\}$ by $I_p(k) = [(8k-8)\delta_p, 8k\delta_p]$ and $I_p(-k) = [-8k\delta_p, -(8k-8)\delta_p]$ for all k, $0 < k \leqslant p+1$. In terms of this function associating integers in A_p with subintervals of $[-1, 1]$, our goal is to make the solution y of Eq. (1) have the following properties:

(i) If y enters the $(s_1 \cdots s_p)$-rectangle from the left with a value in the middle one quarter of $I_p(k)$, then it leaves the $(s_1 \cdots s_p)$-rectangle from the right with a value in the middle one quarter of $I_p(w_G(s_1 \cdots s_p, k))$.

(ii) If y enters from the left in the middle three quarters of $I_p(k)$, then it also leaves from the right in the middle three quarters of $I_p(w_G(s_1 \cdots s_p, k))$. (This is to ensure that approximate solutions to y will not have too big an error.)

First, for Type 1 w_G function, f on the $(s_1 \cdots s_p)$-rectangle is defined as a combination of two upward boxes. Let $x_1 = 0.s_1 \cdots s_p$, $x_2 = x_1 + 2^{-(p+1)}$, $x_3 = x_1 + 2^{-p}$. Also let $z_1 = -\delta_p$, $z_2 = (8(p+2-m)-3)\delta_p$, $z_3 = -(8(p+1-m)+1)\delta_p$, $z_4 = 5\delta_p$. Then f on $[x_1, x_3] \times [-1, 1]$ (the $(s_1 \cdots s_p)$-rectangle) is defined as follows:

(i) f on $[x_1, x_2] \times [z_1, z_2]$ is an upward box with parameters $h_1 = h_2 = 2\delta_p$,

(ii) f on $[x_2, x_3] \times [z_3, z_4]$ is an upward box with parameters $h_1 = h_2 = 2\delta_p$, and

(iii) f is 0 elsewhere on $[x_1, x_3] \times [-1, 1]$.

Figure 2a shows the definition of f on the $(s_1 \cdots s_p)$-rectangle and its effect on the solution y of Eq. (1).

For the function w_G of Type 2, 3, or 4, the corresponding function f is similarly defined. For Type 2 w_G function, f is also a combination of two upward boxes. They are

(i) on $[x_1, x_2] \times [z_1, z_2]$, an upward box with paremeters $h_1 = h_2 = 2\delta_p$ (same as Type 1), and

(ii) on $[x_2, x_3] \times [z_3, -3\delta_p]$, an upward box with parameters $h_1 = h_2 = 2\delta_p$.

For Type 3 w_G function, the corresponding function f is the combination of 2 downward boxes. They are

(i) on $[x_1, x_2] \times [-(8(p+2-m)-5)\delta_p, \delta_p]$, a downward box with parameters $h_1 = h_2 = 2\delta_p$, and

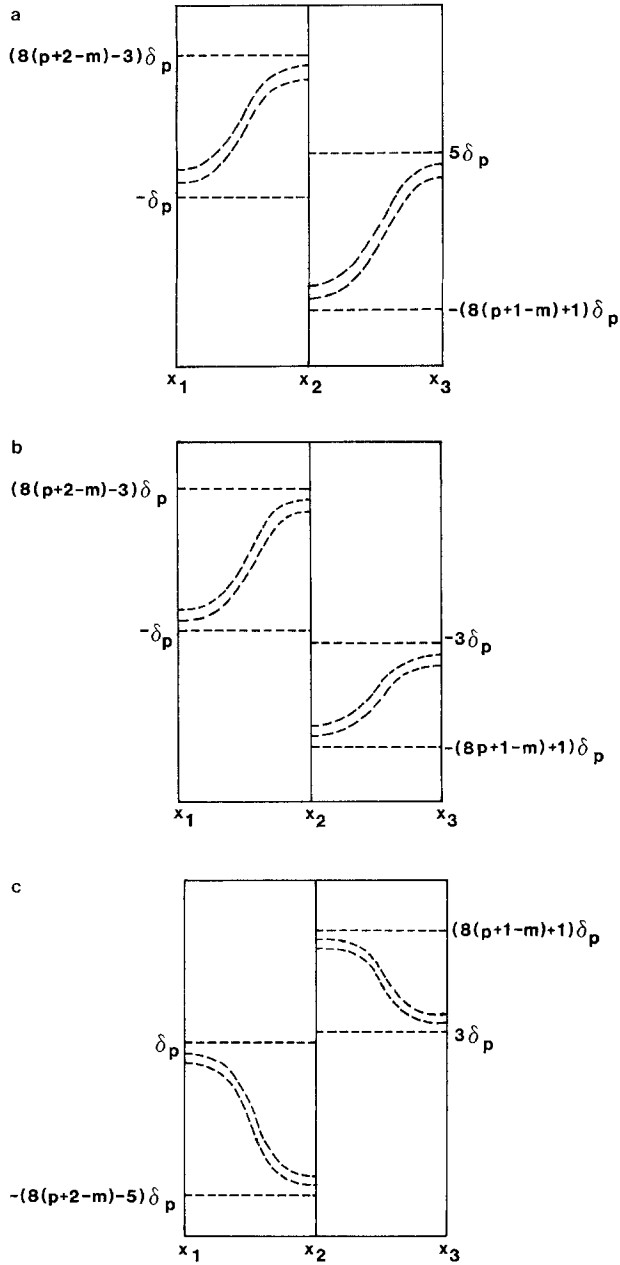

FIG. 2. (a) A type 1 box: an $s_1 \cdots s_p$-box associated with a Type 1 w_G function, with two upward boxes and $h_1 = h_2 = 2\delta_p$.

(b). A type 2 box: 2 upward boxes with $h_1 = h_2 = 2\delta_p$.

(c). A Type 3 box: 2 downward boxes with $h_1 = h_2 = 2\delta_p$.

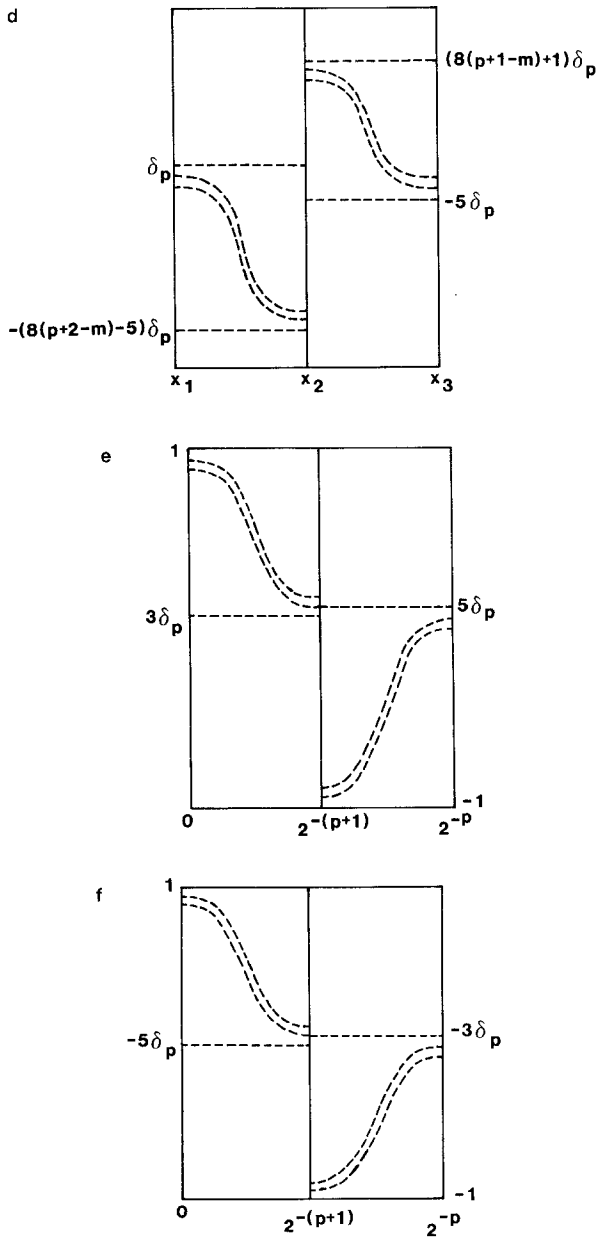

FIG. 2. (d) A Type 4 box: 2 downward boxes with $h_1 = h_2 = 2\delta_p$.

(e) The 0^p-box when $G(0^p)$ is TRUE: an upward box and a downward box with $h_1 = h_2 = 2\delta_p$.

(f) The 0^p-box when $G(0^p)$ is FALSE: an upward box and a down ward box with $h_1 = h_2 = 2\delta_p$.

(ii) on $[x_2, x_3] \times [3\delta_p, (8(p+1-m)+1)\,\delta_p]$, a downward box with parameters $h_1 = h_2 = 2\delta_p$.

For Type 4 w_G function, the corresponding f is similar to that of the Type 3 w_G, except that the second downward box is on the rectangle $[x_2, x_3] \times [-5\delta_p, (8(p+1-m)+1)\,\delta_p]$. These definitions and their effect on the solution y of Eq. (1) are shown in Figs. 2b–d.

Finally, we need to define f on the rectangle $[0, 2^{-p}] \times [-1, 1]$. Here our goal is to make the solution y of Eq. (1) to satisfy the following condition: if y enters the rectangle from left (at 0) with a value $y(0) \in [-1+4\delta_p, 1-4\delta_p]$, then it leaves the rectangle at 2^{-p} with a value $y(2^{-p}) \in [3\delta_p, 5\delta_p]$ if $G(0^p)$ is TRUE, and $y(2^{-p}) \in [-5\delta_p, -3\delta_p]$ if $G(0^p)$ is FALSE. The function f on $[0, 2^{-p}] \times [-1, 1]$ is a combination of an upward box and a downward box. Assume that $G(0^p)$ is TRUE. Then the downward box is the rectangle $[0, 2^{-(p+1)}] \times [3\delta_p, 1]$ with parameters $h_1 = h_2 = 2\delta_p$ and the upward box is the rectangle $[2^{-(p+1)}, 2^{-p}] \times [-1, 5\delta_p]$ with parameters $h_1 = h_2 = 2\delta_p$. Assume that $G(0^p)$ is FALSE. Then, the downward box is the rectangle $[0, 2^{-(p+1)}] \times [-5\delta_p, 1]$, and the upward box is the rectange $[2^{-(p+1)}, 2^{-p}] \times [-1, -3\delta_p]$, and both boxes have parameters $h_1 = h_2 = 2\delta_p$. These two types of functions are shown in Figs. 2e–f.

We first check that our claim that the solution y of Eq. (1) behaves like v_G is correct.

LEMMA 5.4. *Let f be defined as above (from δ_p and G). Then*

$$\max\{|f(x,z)|: 0 \leqslant x \leqslant 1, -1 \leqslant z \leqslant 1\} \leqslant 2^{p+3}.$$

Proof. The maximum value of f is achieved in an upward or a downward box. If an upward box has width a, height b, and two parameters h_1 and h_2, then the maximum value of f on the box is $2(b - 2h_1 - h_2)/a$. Since all our boxes are of width $2^{-(p+1)}$ and height < 2, we have $\max f \leqslant 2^{p+3}$. ∎

Let $(\frac{1}{4})\,I_p(k)$ denote the middle one quarter of the interval $I_p(k)$, and $(\frac{3}{4})\,I_p(k)$ the middle three quarters. That is, if $0 < k \leqslant p+1$, $(\frac{1}{4})\,I_p(k) = [(8k-5)\,\delta_p, (8k-3)\,\delta_p]$ and $(\frac{3}{4})\,I_p(k) = [(8k-7)\,\delta_p, (8k-1)\,\delta_p]$; and $(\frac{1}{4})\,I_p(-k) = [-(8k-3)\,\delta_p, -(8k-5)\,\delta_p]$ and $(\frac{3}{4})\,I_p(-k) = [-(8k-1)\,\delta_p, -(8k-7)\,\delta_p]$.

LEMMA 5.5. *Let $f = f_G$ be defined as above. Then the equation $y'(x) = f(x, y)$ has a unique solution y on $[0, 1]$ for any initial value $y(0) \in [-1, 1]$. Furthermore,*

(a) *for any $s_1 \cdots s_p \in \{0,1\}^p - \{0^p\}$, and any $k \in A_p$, $y(0.s_1 \cdots s_p) \in (\frac{1}{4})\,I_p(k)$ implies $y(0.s_1 \cdots s_p + 2^{-p}) \in (\frac{1}{4})\,I_p(w_G(s_1 \cdots s_p, k))$, and*

$y(0.s_1 \cdots s_p) \in (\frac{3}{4})\, I_p(k)$ *implies* $y(0.s_1 \cdots s_p + 2^{-p}) \in (\frac{3}{4})\, I_p(w_G(s_1 \cdots s_p, k))$, *and*

(b) $y(0) \in [-1 + 4\delta_p,\ 1 - 4\delta_p]$ *implies* $y(2^{-p}) \in (\frac{1}{4})\, I_p(j)$ *with* $j = 1$ *if* $G(0^p)$ *is* TRUE *and* $j = -1$ *if* $G(0^p)$ *is* FALSE.

Proof. The existence and uniqueness of the solution y is guaranteed by the Lipschitz condition for f. Note that $|f(x, y)| \leqslant 2^{p+3}$, and from the definition of the upward and downward boxes, f satisfies the Lipschitz condition everywhere with the Lipschitz constant $2^{p+3} \cdot (2\delta_p)^{-1}$.

Note that an upward box $[x_1, x_2] \times [z_1, z_2]$ with parameters h_1 and h_2 has the solution y with the following properties:

$$y(x_1) \in [z_1, z_1 + h_1] \Rightarrow y(x_2) \in [z_1, z_2 - h_1 - h_2],$$

$$y(x_1) \in [z_1 + h_1,\ z_1 + h_1 + h_2] \Rightarrow y(x_2) \in [z_2 - h_1 - h_2, z_2 - h_1],$$

and

$$y(x_1) \in [z_1 + h_1 + h_2, z_2] \Rightarrow y(x_2) \in [z_2 - h_1, z_2].$$

Similarly, a downward box with the same parameters has the solution y with the following properties:

$$y(x_1) \in [z_1, z_2 - h_1 - h_2] \Rightarrow y(x_2) \in [z_1, z_1 + h_1],$$

$$y(x_1) \in [z_2 - h_1 - h_2, z_2 - h_1] \Rightarrow y(x_2) \in [z_1 + h_1, z_1 + h_1 + h_2],$$

and

$$y(x_1) \in [z_2 - h_1, z_2] \Rightarrow y(x_2) \in [z_1 + h_1 + h_2, z_2].$$

From these properties and the definition of f, we can check that (a) and (b) are true. In the following, we check condition (a) with Type 1 function w_G here, and leave other cases to the reader.

Let $x_1 = 0.s_1 \cdots s_p$, $x_2 = x_1 + 2^{-(p+1)}$ and $x_3 = x_1 + 2^{-p}$. From the definition of f (cf. Fig. 2a), we have $y(x_1) \in [\delta_p, 3\delta_p] \Rightarrow y(x_2) \in [(8k - 7)\,\delta_p, (8k - 5)\,\delta_p]$, where $k = p + 2 - m$, and $y(x_1) \in [3\delta_p, (8k - 3)\,\delta_p] \Rightarrow y(x_2) \in [(8k - 5)\,\delta_p, (8k - 3)\,\delta_p]$. We note that $m \leqslant p$ and so $k \geqslant 2$. That is, $(8k - 7)\,\delta_p > 5\delta_p$. This implies $y(x_3) = y(x_2)$ if $y(x_2) \geqslant (8k - 7)\,\delta_p$. So, $y(x_1) \in (\frac{1}{4})\, I_p(i) \Rightarrow y(x_3) \in (\frac{1}{4})\, I_p(k)$, for all i, $1 \leqslant i \leqslant k$. Also, $y(x_1) \in (\frac{3}{4})\, I_p(i) \Rightarrow y(x_3) \in (\frac{3}{4})\, I_p(k)$, for all i, $1 \leqslant i \leqslant k$.

Next consider i with $-1 \geqslant i \geqslant -(p + 1 - m)$. We note that $y(x_1) = y(x_2)$ for all $y(x_1) \in (\frac{3}{4})\, I_p(i)$, $-1 \geqslant i \geqslant -(p + 1 - m)$. Furthermore, $y(x_2) \in (\frac{1}{4})\, I_p(i) \Rightarrow y(x_3) \in (\frac{1}{4})\, I_p(1) = [3\delta_p, 5\delta_p]$, and $y(x_2) \in (\frac{3}{4})\, I_p(i) \Rightarrow y(x_3) \in [\delta_p, 5\delta_p] \subseteq (\frac{3}{4})\, I_p(1)$, for all i, $-1 \geqslant i \geqslant -(p + 1 - m)$.

Finally, if $i > p + 2 - m$ or $i < -(p + 1 - m)$, $y(x_1) \in (\frac{3}{4})\, I_p(i) \Rightarrow y(x_3) = y(x_2) = y(x_1)$. This completes the proof of (a) with the case that $w_G(s_1 \cdots s_p, k)$ is of Type 1. ∎

LEMMA 5.6. *Let y be the solution of the equation $y'(x) = f(x, y)$ with respect to the function f constructed as above, with the initial value $y(0) \in [-1 + 4\delta_p, 1 - 4\delta_p]$. Then $y(1) \geqslant 3\delta_p$ if G is* TRUE *and $y(1) \leqslant -3\delta_p$ if G is* FALSE.

Proof. Obvious from Lemmas 5.2, 5.3, and 5.5. ∎

Other properties of the function f are summarized in Lemmas 5.7 and 5.8.

LEMMA 5.7. *For a fixed polynomial β, there is a polynomial function α such that for any formula G with p quantifiers and $\delta_p = 2^{-\beta(p)}$, the function f_G can be computed in time $\alpha(n + p)$; i.e., there is an oracle machine M such that for any two functions ϕ and ψ binary converging to real numbers x and $y \in [0, 1]$, respectively, $M^{\phi,\psi}(n)$ halts in $\alpha(n + p)$ moves and $|M^{\phi,\psi}(n) - f_G(x, y)| \leqslant 2^{-n}$.*

Proof. Since w_G is polynomial time computable, and since the evaluation of $G(s_1 \cdots s_p)$ is polynomial time computable, the types of the function f on each $(s_1 \cdots s_p)$-rectangle can be determined in $\gamma(p)$ moves for some polynomial γ. To compute $f(x, y)$, then, we need simply to find the $(s_1 \cdots s_p)$-rectangle which contains (x, y), to figure out what types of boxes (x, y) is in, and to compute according to the definitions of the upward and downward boxes. ∎

LEMMA 5.8. *There is a set $E \subseteq [0, 1] \times [-1, 1]$ such that*

(i) *for any solution y of the equation $y'(x) = f(x, y(x))$ with initial value $y(0) \in [-1 + 4\delta_p, 1 - 4\delta_p]$, the set $\{(x, z)\colon 0 \leqslant x \leqslant 1, |z - y(x)| \leqslant 2\delta_p\}$ is contained in E, and*

(ii) *f satisfies the right Lipschitz condition on E for any right Lipschitz constant $L > 0$.*

Proof. For each upward box $[x_1, x_2] \times [z_1, z_2]$ with parameters h_1 and h_2, let $E \cap [x_1, x_2] \times [z_1, z_2] = [x_1, x_2] \times [z_1 + h_1, z_2]$. For each downward box $[x_1, x_2] \times [z_1, z_2]$ with parameters h_1 and h_2, let $E \cap [x_1, x_2] \times [z_1, z_2] = [x_1, x_2] \times [z_1, z_2 - h_1]$. In addition, let all zero areas in $[0, 1] \times [-1, 1]$ be in E. We observe that for any (x, y), $0 \leqslant x \leqslant 1$, $y \in (\frac{3}{4}) I_p(k)$ for any $k \in A_p$, $(x, y) \in E$. Also by Lemma 5.5. $y(0) \in [-1 + 4\delta_p, 1 - 4\delta_p]$ implies $y(j \cdot 2^{-p}) \in (\frac{1}{4}) I_p(k)$ for some k, for all $j = 1, \ldots, 2^p$. Furthermore, if $y(j \cdot 2^{-p}) \neq y((j + 1) \cdot 2^{-p})$ for some $j = 0, \ldots, 2^p - 1$, then $\{(x, y(x))\colon j \cdot 2^{-p} \leqslant x \leqslant (j + 1) \cdot 2^{-p}\}$ is contained either in the upper part of an upward box (i.e., $z_1 + h_1 + h_2 \leqslant y(x) \leqslant z_2$ if the box is $[x_1, x_2] \times [z_1, z_2]$) or the lower part of a downward box (i.e., $z_1 \leqslant y(x) \leqslant z_2 - h_1 - h_2$). Thus the set $\{(x, z)\colon 0 \leqslant x \leqslant 1, |z - y(x)| \leqslant 2\delta_p\}$ is contained in E because we have $h_1 = 2\delta_p$ for all upward and downward boxes. Thus (i) is proved.

To see that (ii) is true, we note that for any (x, z_1), (x, z_2) in $[0, 1] \times [-1, 1]$ with $z_2 > z_1$, either the line segment connecting them does not lie entirely in E because it crosses over the lower interpolation area of an upward box or the upper interpolation area of a downward box, or $f(x, z_2) \leqslant f(x, z_1)$. This observation completes the proof of (ii). ∎

5.3. *Proof of Theorem* 3, (a) ⇒ (b)

We assume a fixed coding scheme for quantified Boolean formulas such that there is a polynomial time algorithm to determine whether a given binary string encodes a quantified Boolean formula in the normal form. Without loss of generality, we assume that the empty string does not encode any formula.

We divide the interval $[0, 1]$ into infinitely many subintervals such that for each $k \geqslant 1$, there are 2^k many subintervals of length 2^{-2k}. That is, each binary string $s = s_1 \cdots s_k$ of length k is identified with the subinterval $J_s = [\text{left}_s, \text{right}_s]$, where $\text{left}_s = 1 - 2^{-(k-1)} + (0.s)\, 2^{-k}$ and $\text{right}_s = \text{left}_s + 2^{-2k}$. ($0.s$ is the dyadic rational whose binary representation is $0.s$.)

The function f on each rectangle $J_s \times [-1, 1]$ is defined as follows.

Case 1. The string s does not encode a formula in the normal form. Then, f is 0 on $J_s \times [-1, 1]$.

Case 2. The string s encodes a formula G_s in its normal form. Let the length of s be k. Then, G_s has at most k quantifiers. Let $\delta_s = 2^{-(k+5)}$ and let f_s be the function on $[0, 1] \times [-1, 1]$ constructed from G_s and δ_s as described in Section 5.2. Note that $\delta_s \cdot (8k + 12) < 1$.

Let $\beta(k) = (k^2 + 5k - 4)/2$. The rectangle $J_s \times [-1, 1]$ is further divided into $2^{\beta(k)+1}$ many subrectangles: $J_s \times [i \cdot 2^{-\beta(k)}, (i+1) \cdot 2^{-\beta(k)}]$, $i = -2^{\beta(k)}, \ldots, 2^{\beta(k)} - 1$. On each rectangle $J_s \times [i \cdot 2^{-\beta(k)}, (i+1) \cdot 2^{-\beta(k)}]$, called the (s, i)-box, f is defined by a linear transformation from f_s: let $(x, z) \in (s, i)$-box; $f(x, z) = 2^{-(\beta(k)-2k+1)} f_s(\tilde{x}, \tilde{z})$, where $\tilde{x} = 2^{2k}(x - \text{left}_s)$ and $\tilde{z} = -1 + 2^{\beta(k)+1} \cdot (z - i \cdot 2^{-\beta(k)})$.

Assume that y_s is the solution of the equation $y'(x) = f_s(x, y(x))$ on $[0, 1]$ with initial value $y_s(0) \in [-1, 1]$. Then $y(x) = i \cdot 2^{-\beta(k)} + 2^{-(\beta(k)+1)}(y_s(\tilde{x}) + 1)$, where $\tilde{x} = 2^{2k}(x - \text{left}_s)$, is the solution of the equation $y'(x) = f(x, y(x))$ on J_s with initial value $y(\text{left}_s) = i \cdot 2^{-\beta(k)} + 2^{-(\beta(k)+1)}(y_s(0) + 1)$, because $y'(x) = 2^{-(\beta(k)+1)} \cdot y_s'(\tilde{x}) \cdot (d\tilde{x}/dx) = 2^{-(\beta(k)-2k+1)} f_s(\tilde{x}, y_s(\tilde{x})) = f(x, y(x))$. Furthermore, $y(\text{right}_s) \geqslant (2i + 1) \cdot 2^{-(\beta(k)+1)} + 3\delta_s'$ if G_s is TRUE, and $y(\text{right}_s) \leqslant (2i + 1) \cdot 2^{-(\beta(k)+1)} - 3\delta_s'$ if G is FALSE, where $\delta_s' = 2^{-(\beta(k)+1)} \cdot \delta_s = 2^{-(\beta(k)+k+6)}$.

We check that f satisfies all the conditions required in Theorem 3(a).

LEMMA 5.9. *f is polynomial time computable.*

Proof. First note that $\max\{|f(x,z)|: x \in J_s, z \in [-1,1]\} \leqslant 2^{-(\beta(k)-2k+1)} \cdot \max\{f_s(x,z): x \in [0,1], z \in [-1,1]\} \leqslant 2^{-(\beta(k)-2k+1)} \cdot 2^{k+3} = 2^{-(\beta(k)-3k-2)} \leqslant 2^{-k}$ if $k =$ length of $s \geqslant 5$ (by Lemma 5.4).

Assume that γ is the time bound function for f_s as shown in Lemma 5.7. The following algorithm computes f.

(Algorithm)
Input: real numbers x, z.
Output precision: n.

Begin
get a dyadic rational s such that $|s-x| \leqslant 2^{-(\gamma(2n)+2n)}$;
if $s \geqslant 1-2^{-(n+1)}$ **and** $n \geqslant 5$ **then** output 0 **and stop**
else begin
determine the integer $k \leqslant n$ such that
$1-2^{-k} \leqslant s < 1-2^{-(k+1)}$;
let u be the substring of the binary representation of
s from $(k+1)$st bit to $(2k)$th bit;
if u does not encode a formula in the normal form
then output 0
else begin
get a dyadic rational t such that
$|t-z| \leqslant 2^{-(\gamma(n+k)+\beta(k)+1)}$;
let i be the greatest integer satisfying
$i \cdot 2^{-\beta(k)} \leqslant t$;
$\tilde{s} := 2^{2k} \cdot (s - \text{left}_u)$;
$\tilde{t} := -1 + 2^{\beta(k)+1}(t - i \cdot 2^{-\beta(k)})$;
get a dyadic rational v such that
$|v - f_u(\tilde{s}, \tilde{t})| \leqslant 2^{-n}$;
let w be a dyadic rational of length n such that
$|w - 2^{-(\beta(k)-2k+1)} \cdot v|$ is minimum;
output w
end {inner else}
end {outer else}
End.

To see that the above algorithm computes f correctly, we first note that if $s \geqslant 1-2^{-(n+1)}$ then $|f(x,z)| \leqslant 2^{-n}$, and so the output 0 is correct within an error 2^{-n}. Next, assume that $s \leqslant 1-2^{-(n+1)}$. Then, (s,t) is in the (u,i)-box, and $f(s,t) = f_u(\tilde{s},\tilde{t}) \cdot 2^{-(\beta(k)-2k+1)}$. Therefore, $|w - f(s,t)| \leqslant 2^{-n}$. Now, if (x,z) is also in the (u,i)-box, then $|f(s,t) - f(x,z)| \leqslant 2^{-(n+k)}$ because f_u is polynomial time computable. If, on the other hand, (x,z) is not in the (u,i)-box, then both (s,t) and (x,z) must be very close to the boundary of the (u,i)-box. Since we have chosen δ_u to be small enough (i.e.,

$\delta_u \cdot (8k+12) < 1$), f_u has value 0 on the boundary of the (u, i)-box, and is linearly continuous at the small neighborhood of the boundary. This shows that both $f(s, t)$ and $f(x, z)$ are close to zero. We leave the detailed checking to the reader.

The polynomial time computability of the above algorithm follows immediately from the polynomial time computability of f_u and the fact that we can decode a string u in polynomial time to check whether it represents a Boolean formula in the normal form. ∎

LEMMA 5.10. *Equation* (1) *with respect to this f has a unique solution y. There is a set* $E \subseteq [0, 1] \times [-1, 1]$, *and a polynomial* α, *such that*

(i) *f satisfies the right Lipschitz condition on E with the right Lipschitz constant* $L = 1$, *and*

(ii) $\{(x, z)\colon 0 \leqslant x \leqslant 1 - 2^{-k}, |z - y(x)| \leqslant 2^{-\alpha(k)}\} \subseteq E$, *for all* $k \geqslant 1$.

Proof. For each string s of length k which encodes a formula G_s in its normal form, let $E_s \subseteq [0, 1] \times [-1, 1]$ be the set defined in Lemma 5.8 with respect to f_s. For each s and each $i = -2^{\beta(k)}, \ldots, 2^{\beta(k)} - 1$, let $E_{s,i} = \{(\text{left}_s + 2^{-2k} \cdot x, \quad i \cdot 2^{-\beta(k)} + 2^{-(\beta(k)+1)}(z+1))\colon \ (x, z) \in E_s\}$ and $E = (\bigcup \{E_{s,i}\colon s \text{ encodes a formula } G_s \text{ and } -2^{\beta(k)} \leqslant i \leqslant 2^{\beta(k)} - 1\}) \cup (\bigcup \{J_s \times [-1, 1]\colon s \text{ does not encode a formula}\})$. We claim that E satisfies (i) and (ii).

(i) For any $(x, z_1), (x, z_2) \in [0, 1] \times [-1, 1]$, with $z_2 > z_1$, if the line segment connecting them lies entirely in E and if $x \in J_s$ with s encoding a formula G, then either it lies in one (s, i)-box or it lies in two adjacent (s, i)-boxes. For the first case, we have $f(x, z_2) \leqslant f(x, z_1)$ since they are the images of a liear transformation of $f_s(\tilde{x}, \tilde{z}_2) \leqslant f_s(\tilde{x}, \tilde{z}_1)$. For the second case, we know that there is a point (x, z_3) such that $z_2 > z_3 > z_1$ and the line segment from (x, z_1) to (x, z_3) lies in one (s, i)-box and the line segment from (x, z_3) to (x, z_2) lies in another (s, j)-box. So, from the first case, we have $f(x, z_2) \leqslant f(x, z_3) \leqslant f(x, z_1)$. Therefore, f satisfies the right Lipschitz condition on E.

(ii) Let s be a string encoding a formula, and length $(s) = k$. As we observed before, the linear transformation $y(x) = i \cdot 2^{-\beta(k)} + 2^{-(\beta(k)+1)} \cdot (y_s(\tilde{x}) + 1)$ with $\tilde{x} = 2^{2k}(x - \text{left}_s)$ maps the solution y_s of $y'(x) = f_s(x, y(x))$ to the solution y of $y'(x) = f(x, y(x))$. In particular, if $y(\text{left}_s) \in [i \cdot 2^{-\beta(k)} + 4\delta_s', \ (i+1) \cdot 2^{-\beta(k)} - 4\delta_s']$ then $y_s(0) \in [-1 + 4\delta_s, \ 1 - 4\delta_s]$, which implies $\{(\tilde{x}, \tilde{z})\colon \tilde{x} \in [0, 1], |\tilde{z} - y_s(\tilde{x})| \leqslant 2\delta_s\} \subseteq E_s$, and hence $\{(x, z)\colon x \in J_s, \ |z - y(x)| \leqslant 2\delta_s'\} \subseteq E_{s,i} \subseteq E$. So, we need only to check that the solution y of Eq. (1) always enters an (s, i)-box in $[i \cdot 2^{-\beta(k)} + 4\delta_s', \ (i+1) \cdot 2^{-\beta(k)} - 4\delta_s']$. We note that $\delta_s = 2^{-(k+5)} < (8k+12)^{-1}$ and hence $(16k+24)\,\delta_s' < 2^{-\beta(k)}$. Also, the solution y always leaves an (s, i)-box in the middle $(16k+16)\,\delta_s'$ area, provided it enters the (s, i)-box away from the top

and bottom boundaries with distance $> 4\delta'_s$. This means that for any $k \geqslant 1$, if y enters the $(0^k, i)$-box in $[i \cdot 2^{-\beta(k)} + 4\delta'_s,\ (i+1) \cdot 2^{-\beta(k)} - 4\delta'_s]$, then it does so for every (s, i)-box with length of $s = k$. So we are left with the case of the $(0^k, i)$-box.

We observe that for any t of length $k-1$, $\delta'_t = 2^{-(\beta(k-1)+(k-1)+6)}$. Note that $\beta(k-1)+(k-1)+6 = ((k-1)^2 + 5(k-1)-4)/2 + k + 5 = (k^2+5k+2)/2 = \beta(k)+3$. So, $8\delta'_t = 2^{-\beta(k)} =$ the height of the $(0^k, i)$-box. Now, from Lemma 5.5, y_t always leaves $[0,1] \times [-1,1]$ at 1 with $y_t \in (\frac{1}{4}) I_{k-1}(j)$ for some j. That means y leaves a (t, h)-box in the middle one quarter of an $8\delta'_t$-interval; or, y enters $(0^k, i)$-box in the middle one quarter of $[i \cdot 2^{-\beta(k)},\ (i+2)\,2^{-\beta(k)}]$. This completes our proof. ∎

Finally, we prove that if y is polynomial time computable, then QBF $\in P$, and hence $P =$ PSPACE.

LEMMA 5.11. *Assume that y is polynomial time computable. Then,* QBF $\in P$.

Proof. For any given formula G, write it in its normal form and encode it, in our fixed coding scheme, into the binary string s. Assume that the length of s is k. Let $\delta'_s = 2^{-(\beta(k)+k+6)}$. Compute $y(\text{right}_s)$ and get a dyadic rational t of length $\beta(k)+k+6$ such that $|t - y(\text{right}_s)| \leqslant 2^{-(\beta(k)+k+6)}$. Since y leaves an (s, i)-box in the middle $(16k+16)\delta'_s$ area, we have $|y(\text{right}_s) - (2i+1) \cdot 2^{-(\beta(k)+1)}| \leqslant (8k+8)\delta'_s$. This implies $|t - (2i+1) \cdot 2^{-(\beta(k)+1)}| \leqslant (8k+9)\delta'_s$ for some i. Furthermore, this i can be found easily from t.

From Lemma 5.6, $y(\text{right}_s) > (2i+1) \cdot 2^{-(\beta(k)+1)} + 3\delta'_s$ if G is TRUE and $y(\text{right}_s) < (2i+1) \cdot 2^{-(\beta(k)+1)} - 3\delta'_s$ if G is FALSE. Since $|t - y(\text{right}_s)| \leqslant \delta'_s$, we have that $|t - (2i+1) \cdot 2^{-(\beta(k)+1)}| > 2\delta'_s$. Therefore, from t and i we can determine whether G is TRUE: $t > (2i+1) \cdot 2^{-(\beta(k)+1)}$ if and only if G is TRUE. ∎

5.4. *Proof of Theorem* 4, (a) ⇒ (b)

The proof of direction (a) ⇒ (b) of Theorem 4 is similar to that of Theorem 3. Let $A \subseteq \{0\}^*$ be any set in PSPACE$_s$. Since QBF is PSPACE-complete, there is a polynomial time computable function ϕ: $0^* \to \{0,1\}^*$ such that $0^k \in A$ if and only if $\phi(0^k) \in$ QBF. Without loss of generality, we assume that for each $k \geqslant 0$, $\phi(0^k)$ is a quantified Boolean formula G_k with exactly $\beta(k)$ many quantifiers, where β is a polynomial function satisfying $\beta(k) \geqslant k$ and $\beta(k+1) \geqslant \beta(k) + 4\log_2(\beta(k)) + 10$, for all $k \geqslant 0$. (If G_k has fewer than $\beta(k)$ many quantifiers, we can add dummy ones.)

Similarly to Section 5.3, we divide $[0,1]$ into infinitely many subintervals such that for each integer $k \geqslant 0$, there is an associated interval $J_k = [1 - 2^{-k},\ 1 - 2^{-(k+1)}]$ of length $2^{-(k+1)}$. Furthermore, each rectangle $J_k \times [-1,1]$ is

divided into $2^{\beta(k)+2k+4}$ many subrectangles $J_k \times [i \cdot 2^{-(\beta(k)+2k+3)}, (i+1) \cdot 2^{-(\beta(k)+2k+3)}]$, $i = -2^{\beta(k)+2k+3}, \ldots, 2^{\beta(k)+2k+3} - 1$, each subrectangle called a (k, i)-box. For convenience, let $\text{lower}_{k,i} = i \cdot 2^{-(\beta(k)+2k+3)}$, $\text{upper}_{k,i} = (i+1) \cdot 2^{-(\beta(k)+2k+3)}$ and $\text{middle}_{k,i} = (\text{lower}_{k,i} + \text{upper}_{k,i})/2$.

For each $k \geqslant 0$, let $\delta_k = 2^{-(\beta(k+1)-\beta(k)+4)}$. Define f_k on $[0, 1] \times [-1, 1]$ from G_k and δ_k as constructed in Section 5.2. On each (k, i)-box, f is defined by a linear transformation from f_k: Let $(x, z) \in (k, i)$-box. $f(x, z) = 2^{-(\beta(k)+k+3)} \cdot f_k(\tilde{x}, \tilde{z})$, where $\tilde{x} = 2^{k+1}(x - (1 - 2^{-k}))$, and $\tilde{z} = -1 + 2^{\beta(k)+2k+4}(z - \text{lower}_{k,i})$.

The rest of the proof is similar to that of the proof of (a) $\Rightarrow$ (b) of Theorem 3. We only show the necessary modifications here.

First, the function f defined above is polynomial time computable. The main observation needed here is that $\max\{|f(x, z)|: x \in J_k, -1 \leqslant z \leqslant 1\} \leqslant 2^{-(\beta(k)+k+3)} \cdot \max\{|f(\tilde{x}, \tilde{z})|: 0 \leqslant \tilde{x} \leqslant 1, -1 \leqslant \tilde{z} \leqslant 1\} \leqslant 2^{-(\beta(k)+k+3)} \cdot 2^{\beta(k)+3} \leqslant 2^{-k}$ (by Lemma 5.4). Thus, f has a polynomial modulus function. Together with the uniform polynomial time computability of f_k's, f is computable in polynomial time.

Second, Eq. (1) with respect to f has a unique solution y because f satisfies the Lipschitz condition everywhere. Further, y satisfies the following properties.

(i) y always enters a (k, i)-box in $[\text{lower}_{k,i} + 4\delta'_k, \text{upper}_{k,i} - 4\delta'_k]$, where $\delta'_k = 2^{-(\beta(k)+2k+4)} \cdot \delta_k = 2^{-(\beta(k+1)+2k+8)}$.

(ii) y always leaves a (k, i)-box in the middle one quarter of an interval of length $8\delta'_k$, which is equal to the height of a $(k+1, i)$-box.

(iii) y leaves a (k, i)-box with $y(1 - 2^{-(k+1)}) > \text{middle}_{k,i} + 3\delta'_k$ if G_k is TRUE, and $< \text{middle}_{k,i} - 3\delta'_k$ if G_k is FALSE.

The proof of these results is similar to that in Section 5.3.

Third, for each $k \geqslant 0$, let E_k be the set defined in Lemma 5.8 from f_k. We define, for each i, $-2^{\beta(k)+2k+3} \leqslant i \leqslant 2^{\beta(k)+2k+3} - 1$, $E_{k,i} = \{(1 - 2^{-k} + 2^{-(k+1)} \cdot x, 2^{-(\beta(k)+2k+3)}(i + 2z + 2)): (x, z) \in E_k\}$, and let $E = \bigcup_{k,i} E_{k,i}$. Then, f satisfies the right Lipschitz condition on E with Lipschitz constant 1. Furthermore, the set $\{(x, z): 0 \leqslant x \leqslant 1, |z - y(x)| \leqslant 2\delta'_k\}$ is contained in E because of the properties (i) and (ii) of y listed above.

Finally, we show how to compute the truth value of G_k from $y(1)$.

First, we claim that for each $k \geqslant 0$, we can determine the integer i such that y passes through the (k, i)-box. Furthermore, this can be determined from $y(1)$ in $\gamma(k)$ steps for some polynomial γ. We observe the following facts.

(iv) If y enters a (k, i)-box in $[\text{lower}_{k,i} + 4\delta'_k, \text{upper}_{k,i} - 4\delta'_k]$, then y leaves the (k, i)-box in $[\text{lower}_{k,i} + 8\delta'_k, \text{upper}_{k,i} - 8\delta'_k]$. This is because $(16\beta(k) + 32) \cdot \delta'_k \leqslant$ height of a (k, i)-box, and y must leave the (k, i)-box in the middle $(16\beta(k) + 16)\delta'_k$.

(v) If y enters a (k, i)-box, then $y(x) \in [\text{lower}_{k,i}, \text{upper}_{k,i}]$ for all $x \in [1 - 2^{-(k+1)}, 1]$. This is obvious from the fact that y must leave the (k, i)-box from the right and at the same time it enters some $(k+1, j)$-box of smaller height.

(vi) $8\delta'_k =$ height of a $(k+1, j)$-box.

From (iv)–(vi) we can conclude that $|y(1) - \text{middle}_{k,i}| \leqslant$ height of a (k, i)-box $- 8\delta'_k$, if y passes through the (k, i)-box. This information is sufficient for us to determine the integer i from a dyadic number t with $|t - y(1)| \leqslant \delta'_k = 2^{-(\beta(k+1)+2k+8)}$.

Now, to determine whether G_k is TRUE, we find i and j such that y passes through the (k, i)-box and the $(k+1, j)$-box. Note the $\delta'_k = (8 \cdot n)^{-1} \cdot$ height of a (k, i)-box for some integer n, which fact implies $\text{middle}_{k,i}$ is equal to $\text{lower}_{k+1,m}$ for some m, and $\text{middle}_{k,i} \neq \text{middle}_{k+1,j}$ (see Fig. 3). So, we have that G_k is TRUE if and only if $\text{middle}_{k+1,j} > \text{middle}_{k,i}$. Therefore, $0^k \in A$ can be determined in time $\gamma(k)$ for some polynomial γ. Since A is arbitrary, we have shown (a) $\Rightarrow$ (b) of Theorem 4. ∎

Appendix

We prove Lemmas 5.1–5.3 in this Appendix.

Proof of Lemma 5.1. We prove it by induction on p.

(Initial Step). If G has no quantifier, then, from the definition of v_G, $v_G(\varepsilon)$ is positive if and only if G is TRUE.

(Inductive Step). Let $p > 0$. Assume that for all formulas H of $p - 1$ quantifiers and all $t = t_1 \cdots t_{p-1} \in \{0, 1\}^{p-1} - \{0^{p-1}\}$, $v_H(t) > 0$ if and only

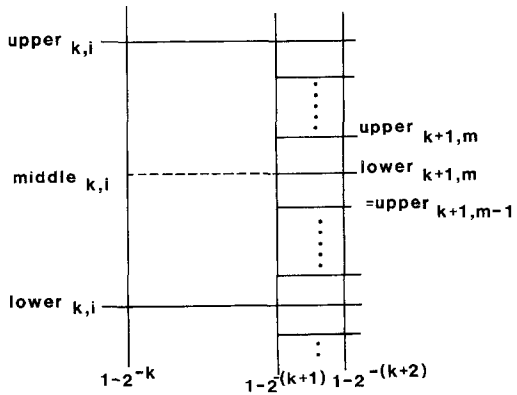

FIG. 3. The relation between a (k, i)-box and adjacent $(k+1, j)$-boxes.

if $H(t_1 \cdots t_k)$ is TRUE, where $k = p - |v_H(t)|$. Let G have p quantifiers, and $s_1 \cdots s_p \neq 0^p$.

Case 1. $s_1 = 0$. Then $v_G(s_1 \cdots s_p) = v_{G(0)}(s_2 \cdots s_p)$. Let $k = p + 1 - |v_G(s_1 \cdots s_p)|$. Then, from the inductive hypothesis, $v_{G(0)}(s_2 \cdots s_p) > 0$ iff $G(0)(s_2 \cdots s_k)$ is TRUE, because $k - 1 = (p-1) + 1 - |v_{G(0)}(s_2 \cdots s_p)|$. Thus, $v_G(s_1 \cdots s_p) > 0$ iff $G(0s_2 \cdots s_k)$ is TRUE.

Case 2. $s_1 = 1$. By the definition of v_G,

$$\begin{aligned} v_G(1s_2 \cdots s_p) &= p + 1 && \text{if } G(0) = \text{TRUE and } Q_1 = \exists, \\ &= -(p+1) && \text{if } G(0) = \text{FALSE and } Q_1 = \forall, \\ &= v_{G(1)}(s_2 \cdots s_p) && \text{otherwise.} \end{aligned}$$

In the first two subcases, $k = 0$. Since $[G(0) = \text{TRUE and } Q_1 = \exists]$ implies $G = \text{TRUE}$, and since $[G(0) = \text{FALSE and } Q_1 = \forall]$ implies $G = \text{FALSE}$, we have $v_G(s_1 \cdots s_p) > 0$ iff G is TRUE. The third subcase is similar to Case 1. ∎

Proof of Lemma 5.2. This lemma can also be proved by induction on p. The initial case is just a special case of Lemma 5.1. Let $p > 0$. Assume that for all formulas H with $p - 1$ quantifiers, $v_H(1^{p-1}) > 0$ iff H is TRUE. Let G have p quantifiers.

Case 1. $Q_1 = \exists$. Then, by the definition of v_G,

$$\begin{aligned} v_G(1^p) &= p + 1 && \text{if } G(0) \text{ is TRUE,} \\ &= v_{G(1)}(1^{p-1}) && \text{if } G(0) \text{ is FALSE.} \end{aligned}$$

That is, $v_G(1^p) > 0$ iff [[$G(0)$ is TRUE] or [$G(0)$ is FALSE but $G(1)$ is TRUE]]. Since $Q_1 = \exists$, G is TRUE iff [$G(0)$ is TRUE or $G(1)$ is TRUE] iff $v_G(1^p) > 0$.

Case 2. $Q_1 = \forall$. The proof is analogous to Case 1. ∎

Proof of Lemma 5.3(a). We prove it by induction on p.

(Initial Step). $p = 1$ implies $s_1 = 1$ and $m = 1$. In this case, we always have $|v_G(0)| \leqslant 1 = p + 1 - m$. So, the initial step is proved.

(Inductive Step). Let $p > 1$. Assume that for all formulas of $p - 1$ quantifiers, (a) is true. Consider a formula G of p quantifiers.

Case 1. $Q_1 = \exists$. We consider subcases as follows.

Subcase 1.1. $s_1 = 0$. Then $\text{pred}(s_1 \cdots s_p) = 0 \cdot \text{pred}(s_2 \cdots s_p)$, and $m(s_2 \cdots s_p) = m - 1$. So,

$$|v_G(\text{pred}(0s_2 \cdots s_p))| > p + 1 - m$$

$$\Rightarrow |v_{G(0)}(\text{pred}(s_2 \cdots s_p))| > p + 1 - m = (p-1) + 1 - (m-1)$$

(by the definition of v_G),

$$\Rightarrow v_G(s_1 \cdots s_p) = v_{G(0)}(s_2 \cdots s_p) \quad \text{(by the definition of } v_G\text{)},$$

$$= v_{G(0)}(\text{pred}(s_2 \cdots s_p)) \quad \text{(by the inductive hypothesis)},$$

$$= v_G(\text{pred}(s_1 \cdots s_p)).$$

Subcase 1.2. $s_1 = 1$ and $m = 1$. That is, $s_2 = \cdots = s_p = 0$. Then, $v_G(\text{pred}(s_1 \cdots s_p)) = v_G(01^{p-1}) = v_{G(0)}(1^{p-1}) \leqslant p$. Therefore (a) is true for this case.

Subcase 1.3. $s_1 = 1$ and $m > 1$. That is, $\text{pred}(s_1 \cdots s_p) = 1 \cdot \text{pred}(s_2 \cdots s_p)$, and $m(s_2 \cdots s_p) = m - 1$.

If $G(0)$ is TRUE, then G is TRUE (because $Q_1 = \exists$), and so both $v_G(s_1 \cdots s_p)$ and $v_G(\text{pred}(s_1 \cdots s_p))$ are defined to be $p + 1$, and (a) is true. If $G(0)$ is FALSE, then the proof is similar to Subcase 1.1.

Case 2. $Q_1 = \forall$. The proof is analogous to Case 1. ∎

Proof of Lemma 5.3(b). Again, the proof is an induction on p.

(Initial Step). $p = 1$ implies $s_1 = 1$ and $m = 1$. From the definition of v_G, we have $v_G(0) = v_{G(0)}(\varepsilon) > 0$ iff $G(0)$ is TRUE. Because $Q_m = Q_1 = \exists$, we have

$$v_G(0) > 0 \Rightarrow G(0) \text{ is TRUE}$$

$$\Rightarrow v_G(1) = p + 1 = 2 = p + 2 - m$$

and

$$v_G(0) < 0 \Rightarrow G(0) \text{ is FALSE}$$

$$\Rightarrow v_G(1) = 1 \quad \text{if } G(1) \text{ is TRUE},$$

$$= -1 \quad \text{if } G(1) \text{ is FALSE}.$$

This proves the initial step.

(Inductive Step). Let $p > 1$. Assume that (b) is true for all formulas of $p - 1$ quantifiers. Let G have p quantifiers. Similarly to the proof of Lemma 5.3(a), we consider the following three cases.

Case 1. $s_1 = 0$. Then $m > 1$, $\text{pred}(s_1 \cdots s_p) = 0 \cdot \text{pred}(s_2 \cdots s_p)$ and $m(s_2 \cdots s_p) = m - 1$. Furthermore, the $(m-1)$st quantifier of $G(0)$ is $\exists$.

From the definition of v_G, $v_G(s_1 \cdots s_p) = v_{G(0)}(s_2 \cdots s_p)$, and $v_G(\text{pred}(s_1 \cdots s_p)) = v_{G(0)}(\text{pred}(s_2 \cdots s_p))$. So,

$$0 < v_G(\text{pred}(s_1 \cdots s_p)) \leqslant p + 1 - m$$
$$\Rightarrow 0 < v_{G(0)}(\text{pred}(s_2 \cdots s_p)) \leqslant (p-1) + 1 - (m-1)$$
$$\Rightarrow v_G(s_1 \cdots s_p) = v_{G(0)}(s_2 \cdots s_p) = (p-1) + 2 - (m-1) = p + 2 - m$$

and

$$-(p + 1 - m) \leqslant v_G(\text{pred}(s_1 \cdots s_p)) < 0$$
$$\Rightarrow -(p + 1 - m) \leqslant v_{G(0)}(\text{pred}(s_2 \cdots s_p)) < 0$$
$$\Rightarrow v_G(s_1 \cdots s_p) = v_G(0)(s_2 \cdots s_p)$$
$$= 1 \qquad \text{if } G(0)(s_2 \cdots s_p) \text{ is TRUE,}$$
$$= -1 \qquad \text{otherwise.}$$

So this case is proved.

Case 2. $s_1 = 1$ and $m = 1$. Then, $s_2 = \cdots = s_p = 0$ and $\text{pred}(s_1 \cdots s_p) = 01^{p-1}$. By Lemma 5.2, $v_G(01^{p-1}) = v_{G(0)}(1^{p-1}) > 0$ iff $G(0)$ is TRUE. Therefore, $v_G(\text{pred}(s_1 \cdots s_p)) > 0 \Rightarrow G(0)$ is TRUE $\Rightarrow v_G(10^{p-1}) = p + 1 = p + 2 + m$ and $v_G(\text{pred}(s_1 \cdots s_p)) < 0 \Rightarrow G(0)$ is FALSE $\Rightarrow v_G(s_1 \cdots s_p) = 1$ if $G(s_1 \cdots s_p)$ is TRUE, and $= -1$ otherwise. This completes the proof of Case 2.

Case 3. $s_1 = 1$ and $m > 1$. Then, $\text{pred}(s_1 \cdots s_p) = 1 \cdot \text{pred}(s_2 \cdots s_p)$, and $m(s_2 \cdots s_p) = m - 1$. Since $p + 1 - m > p$, we know that $0 < v_G(\text{pred}(s_1 \cdots s_p)) \leqslant p + 1 - m < p$ implies $[Q_1 = \exists$ and $G(0)$ is FALSE$]$ or $[Q_1 = \forall$ and $G(0)$ is TRUE$]$. In either case, we have $v_G(\text{pred}(s_1 \cdots s_p)) = v_{G(1)}(\text{pred}(s_2 \cdots s_p))$ and $v_G(s_1 \cdots s_p) = v_{G(1)}(s_2 \cdots s_p)$. Thus, by the inductive hypothesis,

$$0 < v_G(\text{pred}(s_1 \cdots s_p)) \leqslant p + 1 - m$$
$$\Rightarrow 0 < v_{G(1)}(\text{pred}(s_2 \cdots s_p)) \leqslant p + 1 - m$$
$$\Rightarrow v_G(s_1 \cdots s_p) = v_{G(1)}(s_2 \cdots s_p) = (p-1) + 2 - (m-1) = p + 2 - m.$$

Also,

$$-(p + 1 - m) \leqslant v_G(\text{pred}(s_1 \cdots s_p)) < 0$$
$$\Rightarrow -(p + 1 - m) \leqslant v_{G(1)}(\text{pred}(s_2 \cdots s_p)) < 0$$

$$\Rightarrow v_G(s_1 \cdots s_p) = v_{G(1)}(s_2 \cdots s_p)$$
$$= 1 \qquad \text{if } G(1)(s_2 \cdots s_p) \text{ is TRUE,}$$
$$= -1 \qquad \text{otherwise.}$$

This completes the proof of Case 3. ∎

Proof of Lemma 5.3(c) is analogous to that of Lemma 5.3(b).

RECEIVED September 1, 1983; ACCEPTED December 28, 1983

REFERENCES

ABERTH, O. (1971), The failure in computable analysis of a classical existence theorem for differential equations, *Proc. Amer. Math. Soc.* **30**, 151–156.

BOOK, R. V. (1974), Tally languages and complexity classes, *Inform. Contr.* **26**, 186–193.

CLEAVE, J. (1969), The primitive recursive analysis of ordinary differential equations and the complexity of their solutions, *J. Comput. System Sci.* **3**, 447–455.

FREIDMAN, H. (1984), On the computational complexity of maximization and integration, *Adv. in Appl. Math.*, in press.

GAREY, M. R., AND JOHNSON, D. S. (1979), "Computers and Intractability," Freeman, San Francisco.

GRZEGORCZYK, A. (1953), Some classes of recursive functions, *Rozprawy Mat.* **4**, 1–46.

HENRICI, P. (1962), "Discrete Variable Methods in Ordinary Differential Equations," Chap. 1, Wiley, New York.

KNUTH, D. E., AND MOORE, R. W. (1975), An analysis of alpha–beta pruning, *Artificial Intelligence* **6**, 293–326.

KO, K. (1983), On self-reducibility and weak p-selectivity, *J. Comput. System Sci.* **26**, 209–221.

KO, K., AND FRIEDMAN, H. (1982), Computational complexity of real functions, *Theoret. Comput. Sci.* **20**, 323–352.

MEYER, A., AND PATERSON, M. (1979), With what frequency are apparently intractable problems difficult? MIT Tech. Report MIT/LES/TM-126, Cambridge, Massachusetts.

MILLER, W. (1970), Recursive function theory and numerical analysis, *J. Comput. System Sci.* **4**, 465–472.

POUR-EL, M. B., AND RICHARDS, J. (1979), A computable ordinary differential equation which possesses no computable solution, *Ann. Math. Logic* **17**, 61–90.

SCHNORR, C. P. (1976), Optimal algorithms for self-reducible problems, *in* "Third International Colloquium on Automata, Languages and Programming, Edinburgh."

STOCKMEYER, L. J., AND MEYER, A. R. (1973), Word problem requiring exponential time, *in* "Proceedings, 5th Ann. ACM Symposium on Theory of Computing," pp. 1–9.