Characterizing Polynomial Time Computability of Rational and Real Functions

Walid Gomaa
INRIA Nancy Grand-Est Research Centre, France
Faculty of Engineering, Alexandria University, Alexandria, Egypt
walid.gomaa@loria.fr

Recursive analysis was introduced by A. Turing [1936], A. Grzegorczyk [1955], and D. Lacombe [1955]. It is based on a discrete mechanical framework that can be used to model computation over the real numbers. In this context the computational complexity of real functions defined over compact domains has been extensively studied. However, much less have been done for other kinds of real functions. This article is divided into two main parts. The first part investigates polynomial time computability of rational functions and the role of continuity in such computation. On the one hand this is interesting for its own sake. On the other hand it provides insights into polynomial time computability of real functions for the latter, in the sense of recursive analysis, is modeled as approximations of rational computations. The main conclusion of this part is that continuity does not play any role in the efficiency of computing rational functions. The second part defines polynomial time computability of arbitrary real functions, characterizes it, and compares it with the corresponding notion over rational functions. Assuming continuity, the main conclusion is that there is a conceptual difference between polynomial time computation over the rationals and the reals manifested by the fact that there are polynomial time computable rational functions whose extensions to the reals are not polynomial time computable and vice versa.

1 Introduction

Recursive analysis is an approach for investigating computation over the real numbers; it provides a theoretical framework for numerical algorithms. The field was introduced by A. Turing [12], A. Grzegorczyk [7], and D. Lacombe [10]. The computation model is based on the mechanistic view of classical computability theory. Hence, unlike other models of real computation such other C. Moore’s recursive class [11], recursive analysis takes more direct and realistic approach by appealing to the notion of Turing machine and hence physical realizability. A conventional machine equipped with function oracles, called Type II Turing machine, is adopted.

In this article we assume the binary alphabet \{0, 1\}. Hence, in such a context finite strings are interpreted as those rationals with finite binary representation. These are called the dyadic rationals. We denote them \(\mathbb{D}\), and will be assumed throughout the rest of this article as representations of real numbers as indicated by the following definition.

Definition 1. (Cauchy sequence representation of real numbers) Assume \(x \in \mathbb{R}\). Then \(x\) can be represented by a Cauchy function \(\phi_x : \mathbb{N} \to \mathbb{D}\) that converges at a binary rate:

\[
\forall n \in \mathbb{N}: |x - \phi_x(n)| \leq 2^{-n}
\]

(1)

Given \(x \in \mathbb{R}\), let \(CF_x\) denote the class of Cauchy functions that represent \(x\).
The computational complexity of real functions defined over compact domains has been extensively studied (see for example, [9]). However, much less have been done regarding functions over non-compact domains (see foundational work [7][8][10], work investigating the elementary functions and the Grzegorczyk hierarchy [3][6][5], connection with the GPAC model [4], characterization by function algebras [2]). The main crux of the current work is to investigate polynomial time computability of rational and real functions defined over domains with components of the form \([a, \infty)\).

The article is divided into two main parts. The first part investigates polynomial time computability of rational functions and the role of continuity in such computation. On the one hand this is interesting for its own sake. On the other hand it provides insights into polynomial time computability of real functions for the latter, in the sense of recursive analysis, is modeled as approximations of rational computations. The main conclusion of this part is that continuity does not play any role in the efficiency of computing rational functions. There are polynomial time computable rational functions that are discontinuous. Even we can find efficiently computable functions that are ill-behaved from the smoothness perspective, that is, they have arbitrarily large moduli.

The second part defines polynomial time computability of arbitrary real functions, characterizes it for the particular case where the functions are defined over domains with components of the form \([a, \infty)\), and finally compares this notion of computability with the corresponding one over rational functions. Assuming continuity, the main conclusion of this part is that there is a major conceptual difference between polynomial time computation over the rationals and over the reals. This is manifested by the fact that there are continuous polynomial time computable rational functions whose extensions to the reals are not polynomial time computable and conversely, there are rational-preserving polynomial time computable real functions whose restriction to the rationals are not polynomial time computable.

The article is organized as follows. Section 1 is an introduction. Section 2 gives the basic definitions and notions of polynomial time computation over the dyadic numbers along with some results about the features of real computation and its relationship to rational computation. Section 3 gives the basic definitions and notions of polynomial time computation over the reals along with some results about the features of real computation and its relationship to rational computation. Section 4 outlines future research directions to be pursued.

2 Polynomial Time Computability over the Dyadic Numbers

Let \(\Sigma = \{0, 1, -, .\}\) and \(\Gamma = \{00, 01, 10, 11\}\). Define a function \(\tau: \Sigma \to \Gamma\) as follows: \(\tau(0) = 00, \tau(1) = 11, \tau(-) = 01, \tau(.) = 10\). Assume that \(\mathbb{D}\) is the set of dyadic numbers represented in lowest forms with the alphabet \(\Sigma\). Define a function \(\tau^*: \mathbb{D} \to \Gamma^*\) as follows: \(\tau^*(a_0, \ldots, a_n) = \tau(a_0) \cdots \tau(a_n)\). For any \(d \in \mathbb{D}\) let \(\text{len}(d)\) denote the length of the binary string \(\tau^*(d)\).

\textbf{Definition 2} \((P\text{Time})\) (Polynomial time) complexity over \(\mathbb{D}\). Assume a function \(f: \mathbb{D} \to \mathbb{D}\). We say that \(f\) is polynomial time computable if there exists a Turing machine \(M\) such that for every \(d \in \mathbb{D}\) the following holds

\[ M(\tau^*(d)) = \tau^*(f(d)) \]  

and the computation time of \(M\) is bounded by \(p(|\tau^*(d)|)\) for some polynomial function \(p\).

Let \(\mathcal{P}_\mathbb{D}\) denote the class of \(P\text{Time}\) computable dyadic functions. For simplicity in the following discussion we focus for the most part on unary functions.

For ease of readability we will use the notations for intervals to indicate just the dyadics in these intervals. In the following discussion we will need to distinguish between two subclasses of \(\mathcal{P}_\mathbb{D}\). The first class is \(\mathcal{P}_\mathbb{D}[\mathbb{R}] = \{f: [a, \infty) \to \mathbb{D} | f \in \mathcal{P}_\mathbb{D}, a \in \mathbb{D}\}\). The second class is \(\mathcal{P}_\mathbb{D}[\mathbb{H}] = \{f: (a, \infty) \to \mathbb{D} | f \in \mathcal{P}_\mathbb{D}, a \in \mathbb{D}\}\).
Let \( \varepsilon \) Lemma 5. There exists \( D = 56 \) Characterizing Polynomial Time Computability of Rational and Real Functions \( P \in \mathbb{D}; a, b \in \mathbb{D} \). By removing the restriction \( f \in \mathbb{D} \), let \( \mathbb{D}[\mathbb{B}] \) and \( \mathbb{D}[\mathbb{U}] \) denote the resulting classes.

**Definition 3** (Continuous dyadic functions).

1. Assume a function \( f \in \mathbb{D}[\mathbb{B}] \) with domain \([a, \infty)\). We say that \( f \) is continuous if \( f \) has a modulus of continuity, that is if there exists a function \( m: \mathbb{N}^2 \rightarrow \mathbb{N} \) such that for every \( k, n \in \mathbb{N} \) and for every \( x, y \in [a, a + 2^k] \) the following holds:

\[
if \ |x - y| \leq 2^{-m(k, n)}, then \ |f(x) - f(y)| \leq 2^{-n}
\]

(3)

2. Assume a function \( f \in \mathbb{D}[\mathbb{U}] \) with domain \((a, b)\) (other cases can be similarly handled). We say that \( f \) is continuous if \( f \) has a modulus of continuity, that is if there exists a function \( m: \mathbb{N}^2 \rightarrow \mathbb{N} \) such that for every \( k, n \in \mathbb{N} \) and for every \( x, y \in [a + 2^{-k}, b - 2^{-k}] \) the following holds:

\[
if \ |x - y| \leq 2^{-m(k, n)}, then \ |f(x) - f(y)| \leq 2^{-n}
\]

(4)

In the following we will refer to \( k \) as the extension argument and \( n \) as the precision argument.

**Remark 4.**

1. It is clear from Definition 3 that the open domain of a function is divided into a sequence of compact subintervals, each of which corresponds to a fixed \( k \). In the following discussion we will typically need to fix such a compact subinterval, hence to ease the notation we use \( m_k(n) = m(k, n) \) to indicate the modulus of continuity over the fixed subinterval.

2. It is also clear that any continuous dyadic function is bounded over any compact subinterval.

3. As opposed to the case of \( \mathbb{R} \)-computation, computability over \( \mathbb{D} \) does not imply continuity of the underlying function.

Let \( P_{\mathbb{D}}[\mathbb{B}] \) denote the continuous subclass of \( P_{\mathbb{D}} \), similarly for \( P_{\mathbb{D}}[\mathbb{B}, \text{cnt}] \) and \( P_{\mathbb{D}}[\mathbb{U}, \text{cnt}] \). Let \( P_{\mathbb{N}} \) denote the class of \( PTime \) computable \( \mathbb{N} \)-functions. Assume a unary function \( f \in P_{\mathbb{N}} \). Let \( \tilde{f} \in P_{\mathbb{D}}[\mathbb{B}] \) with domain \([0, \infty)\) be an extension of \( f \) defined as follows:

\[
\tilde{f}(x) = f(\lfloor x \rfloor)
\]

(5)

Let \( \mathcal{P} = \{ \tilde{f}: f \in P_{\mathbb{N}} \} \) and let \( \mathbb{D}_0^+ \) denote the set of nonnegative dyadics.

**Lemma 5.** There exists \( f \in P_{\mathbb{D}}[\mathbb{B}, \text{cnt}] \) whose computation time is not bounded by any function in \( \mathcal{P} \). In other words for every \( \tilde{f} \in \mathcal{P} \) the following holds for infinitely many \( d \in \text{dom}(f): \text{len}(f(d)) > \tilde{f}(d) \).

**Proof.** We will construct a function \( f \) such that through an interval \([r, r + 1)\), for \( r \in \mathbb{N} \), the function grows piecewise linear with predetermined breakpoints chosen such that the length of \( f \) grows polynomially in terms of the length of the dyadic input, however, it grows exponentially in terms of the length of the integer part of the input. For \( a \in \mathbb{N} \), let \( \varepsilon_a \) denote the fraction \( 0.01^a \). Let \( d_0 = r \), and let \( k = \min \{ i \in \mathbb{N}: r + 1 \in [0, 2^i) \} \). For every \( j \in \{ 1, \ldots, r \} \) let \( d_j = d_0 + \varepsilon_j \), \( \delta_j = \varepsilon_j - \varepsilon_{j-1} = 2^{-2j} \). The \( d_j \)'s are breakpoints through which the function increases piecewise linearly. Let \( e_0 = d_r + \varepsilon_r \) and for every \( j \in \{ 1, \ldots, r \} \) let \( e_j = e_0 - \varepsilon_j \). The \( e_j \)'s are breakpoints through which the function decreases piecewise linearly, these are needed to maintain continuity. The formal definition of \( f: \mathbb{D}_0^+ \rightarrow \mathbb{D} \) (over every interval
The total length of the two smallest intervals \([r, r + 1]\) is as follows:

\[
\begin{align*}
    f(d) &= \begin{cases} 
        0 & d \in \mathbb{N} \\
        0 & e_0 \leq d \leq r + 1 \\
        2^j & d = d_j, j \in \{1, \ldots, r\} \\
        2^j & d = e_j, j \in \{1, \ldots, r\} \\
        \frac{\delta f(d_{j+1}) + (\delta_{j+1} - \delta) f(d_j)}{\delta_{j+1}} & d_j < d < d_{j+1}, \delta = d - d_j \\
        \frac{\delta f(e_{j+1}) + (\delta_{j+1} - \delta) f(e_j)}{\delta_{j+1}} & e_{j+1} < d < e_j, \delta = e_j - d
    \end{cases}
\end{align*}
\]  

(6)

Note that \(\text{len}(f(d_j))\) is linear in \(\text{len}(d_j)\), similarly for \(\text{len}(f(e_j))\), hence \(f\) is \(\text{PTime}\) computable. Note that \(f(d_r) = 2^r = \Omega(2^{2^{\log(r)}})\), hence \(f\) is not \(\text{PTime}\) computable with respect to \(\mathbb{N}\)-points and therefore its computation is not bounded by any function in \(\mathcal{B}\). By its definition \(f\) is continuous. In the following we will find a modulus function for \(f\). Define a function \(m: \mathbb{N}^2 \to \mathbb{N}\) by \((k, n) \mapsto 3 \cdot 2^k + n\). Let \(\ell_j\) denote the interval \([d_{j-1}, d_j]\) for \(j \in \{1, \ldots, r\}\). Assume \(x, y \in [r, d_r]\) such that \(|x - y| \leq 2^{-m(k,n)}\) (other cases are either trivial or can be handled symmetrically). Note that \(f\) is monotonically increasing piecewise linear over \([r, d_r]\) and the slope of the line in interval \(\ell_j\), where \(j > 1\), can be computed as follows:

\[
f'_j = \frac{f(d_j) - f(d_{j-1})}{\delta_j} = \frac{2^j - 2^{j-1}}{2^{-2j}} = 2^{3j-1}
\]

case 1: \(x, y \in \ell_j\): Note that \(\delta_j = 2^{-2j} \geq \delta_r = 2^{-2r}\). We have

\[
|f(y) - f(x)| = |(y - x) \frac{f(y) - f(x)}{y - x}|
\]

\[
= |y - x| f'_j \\
\leq |y - x| f'_r \\
\leq 2^{-(3 \cdot 2^{k+n})} f'_r \\
\leq 2^{-(3 \cdot 2^{k+n})} 2^{3r-1} \\
\leq 2^{-(n+1)}
\]

case 2: \(x \in \ell_j\) and \(y \in \ell_{j+1}\) where \(1 < j \leq r - 1\):

\[
|f(x) - f(y)| \leq |f(x) - f(d_j)| + |f(d_j) - f(y)| \\
\leq 2^{-(n+1)} + 2^{-(n+1)} \quad \text{(from case 1)} \\
= 2^{-n}
\]

The total length of the two smallest intervals \(\ell_r, \ell_{r-1}\) is \(\delta_r + \delta_{r-1} = 2^{-2r} + 2^{-2(r-1)} > 2^{-2(r-1)} \geq 2^{-3(r+n)} \geq |x - y|\). Hence, it can not happen that \(x \in \ell_i\) and \(y \in \ell_j\) with \(|j - i| > 1\). Therefore, \(m\) is a modulus function for \(f\).

**Lemma 6.** There exists \(f \in \mathcal{D}_{\mathcal{B}}[\mathcal{B}, \text{cnt}]\) that does not have a polynomial modulus with respect to the extension argument \(k\).
**Proof.** Investigating the proof of Lemma 5 it can be easily seen that the function \( f \) defined in that proof satisfies the conclusion of this lemma.

**Remark 7.** By looking again into the proof of Lemma 5 we observe that the apex of the graph of \( f \) can be taken as arbitrarily high as we want by letting \( j \) runs from 1 to \( \alpha(r) \) for any monotonically increasing function \( \alpha \). This indicates that there is no upper bound on the moduli of continuity of the functions in \( P_{\mathbb{B}}[\mathbb{B},\text{cnt}] \).

**Lemma 8.** There exists a function \( f \in P_{\mathbb{B}}[\mathbb{B},\text{cnt}] \) that does not have a polynomial modulus with respect to the precision argument \( n \).

**Proof.** Define a function \( \alpha : \mathbb{N} \to \mathbb{N} \) by

\[
\alpha(i) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } i = 1 \\
\max \{ j \leq i : \lfloor \log_2 \log_2 j \rfloor = \log_2 \log_2 j \} & \text{otherwise} 
\end{cases}
\]

(7)

For every \( i \in \mathbb{N} \) let \( d_i = 1 - 2^{-i} \). Define a function \( f : \mathbb{D}_0^+ \to \mathbb{D} \) as follows:

\[
f(d) = \begin{cases} 
\frac{1}{\log_2 \alpha(i)} & d_0 \leq d \leq d_1 \\
\frac{1}{2^{(j+1)}} \left( 2^{-i} f(d_{i+1}) + (2^{-(i+1)} - \delta) f(d_i) \right) & d_i < d < d_{i+1}, \delta = d - d_i \\
0 & d \geq 1 
\end{cases}
\]

(8)

Then \( f \) is a piecewise linear decreasing function over the interval \([0, 1]\). It decreases very slowly (may even remain constant over long subintervals), however, it eventually reaches 0 at \( d = 1 \), thus it is continuous. Note that for every \( i \), \( \text{len}(d_i) = i \) and by definition \( \text{len}(f(d_i)) = O(\log(i)) \). In addition \( f(d_i) \) is efficiently computable. Hence, \( f \) is \( P\text{Time} \) computable. Finally, we need to show that \( f \) does not have a polynomial modulus with respect to the precision parameter. Let \( \ell_i \) denote the subinterval \([d_{i-1}, d_i]\). Note that there are infinitely many \( \ell_i \)'s over which \( f \) is strictly decreasing. The goal is to compute the slope of the function over such subintervals. Assume such an interval \( \ell_i = [d_{i-1}, d_i] \), then it must be the case that \( i = 2^j \) for some \( j \in \mathbb{N} \).

\[
|\ell_i| = d_i - d_{i-1} \\
= 1 - 2^{-i} - 1 + 2^{-(i-1)} \\
= 2^{-i} = 2^{-2^j}
\]

On the other hand

\[
f(d_{i-1}) - f(d_i) = 2^{-(j-1)} - 2^{-j} \\
= 2^{-j}
\]

Hence, the slope of the line over \( \ell_i \) is

\[
|f'| = \frac{2^{-j}}{2^{-(2^j)}} \\
= \frac{2^{2^j-j}}{j}
\]

which can not be captured by any polynomial function. \( \square \)
Combining the proofs of Lemma 6 and Lemma 8 we have the following.

**Theorem 9.** There exists a function \( f \in \mathcal{P}_B[\mathcal{B}, \text{cut}] \) that does not have a polynomial modulus with respect to both the extension parameter \( k \) and the precision parameter \( n \) (that is if one variable is considered constant the function would not be polynomial in the other).

### 3 Polynomial Time Computability over the Real Numbers

Since continuity is a necessary condition for computation over \( \mathbb{R} \), it will not be mentioned explicitly. So we use the notation \( \mathcal{P}_\mathbb{R} \) to denote continuous PTIME computable \( \mathbb{R} \)-functions. Again we divide into two subcases: \( \mathcal{P}_\mathbb{R}[\mathcal{B}] \) and \( \mathcal{P}_\mathbb{R}[\mathcal{U}] \). In this section we exclusively handle the case \( \mathcal{P}_\mathbb{R}[\mathcal{B}] \). In the case of real functions over compact domains there is only one parameter that controls the computational complexity, namely the precision (the level of approximation required for the output). Given a positive integer \( n \) as an input to the machine computing such a function, then roughly speaking \( n \) is the required length of the output. Over a compact domain there are predetermined lower and upper bounds on the function value, hence the length of the integer part can be considered constant and therefore does not play any role in the complexity. For a detailed discussion about functions over compact domains see [9]. Moving to functions in \( \mathcal{P}_\mathbb{R}[\mathcal{B}] \) the domain and generally the range become unbounded hence an additional parameter, namely the length of the integer part, must now be accounted for in the computational complexity of a function. Unlike the precision parameter which is an explicit input the extension parameter is extracted by the machine by asking the oracle that represents the actual real number input.

Given a Cauchy sequence \( \varphi \) let \( M^\varphi \) denote a Turing machine \( M \) that has access to oracle \( \varphi \).

**Definition 10.** (PTIME complexity over \( \mathbb{R}[\mathcal{B}] \)) Assume a function \( f: [a, \infty) \rightarrow \mathbb{R} \). We say that \( f \) is polynomial time computable if the following conditions hold:  

1. There exists a two-function oracle Turing machine such that for every \( \varphi_a \in CF_a \), \( x \in [a, \infty) \), \( \varphi_x \in CF_x \), and for every \( n \in \mathbb{N} \) the following holds:  
   \[
   |M^{\varphi_a, \varphi_x} (n) - f(x)| \leq 2^{-n} \tag{9}
   \]

2. The computation time of \( M^{\varphi} (n) \) is bounded by \( p(k, n) \) for some polynomial \( p \), where \( k = \min \{ j : x \in [a, a + 2^j] \} \).

**Remark 11.** Note that in the previous definition there was not any computability restrictions over the constant \( a \). Furthermore, we chose to universally quantify over all possible Cauchy sequence representations of \( a \). This avoids the risk of apriori fixing a particular Cauchy sequence which might contain hyper computational information (such as the encoding of the halting set).

**Example 12.** Consider the function \( f: [0, \infty) \rightarrow \mathbb{R} \), defined by \( f(x) = e^x \). Note that \( f(x) = \Omega(2^x) \). Since \( 2^x \upharpoonright \mathbb{N} \) is not polynomial time computable as an \( \mathbb{N} \)-function, it is not either polynomial time computable as an \( \mathbb{R} \)-function. Thus, \( f \) is not polynomial time computable.

**Notation 13.** For any \( x \in \mathbb{R} \), let \( \varphi_x^* \in CF_x \) denote the particular Cauchy function  
\[
\varphi_x^*(n) = \frac{\lfloor 2^n \cdot x \rfloor}{2^n} \tag{10}
\]

**Theorem 14.** (Characterizing \( \mathcal{P}_\mathbb{R}[\mathcal{B}] \)) Assume a function \( f: [a, \infty) \rightarrow \mathbb{R} \). Then \( f \) is polynomial time computable iff there exist two functions \( m : \mathbb{N}^2 \rightarrow \mathbb{N} \) and \( \psi : \mathbb{D} \cap [a, \infty) \times \mathbb{N} \rightarrow \mathbb{D} \) such that...
1. $m$ is a modulus function for $f$ and it is polynomial with respect to both the extension parameter $k$ and the precision parameter $n$, that is, $m(k,n) = (k+n)^b$ for some $b \in \mathbb{N}$.

2. $\psi$ is an approximation function for $f$, that is, for every $d \in \mathbb{D} \cap [a,\infty)$ and every $n \in \mathbb{N}$ the following holds:

$$|\psi(d,n) - f(d)| \leq 2^{-n}$$  

3. $\psi(d,n)$ is computable in time $p(|d|+n)$ for some polynomial $p$.

**Proof.** Fix some $\varphi_a \in CF_a$. The proof is an extension of the proof of Corollary 2.21 in [9]. Assume the existence of $m$ and $\psi$ that satisfy the given conditions. Assume an $f$-input $x \in [a,\infty)$ and let $\varphi_a \in CF_a$. Assume $n \in \mathbb{N}$. Let $M^{\varphi_a}(n)$ be an oracle Turing machine that does the following:

1. let $d_1 = \varphi_a(2)$ and $d_2 = \varphi_a(2)$.
2. from $d_1$ and $d_2$ determine the least $k$ such that $x \in [a,a+2^k]$.
3. let $\alpha = m(k,n+1)$.
4. let $d = \varphi_a(\alpha)$.
5. let $e = \psi(d,n+1)$ and output $e$.

Note that every step of the above procedure can be performed in polynomial time with respect to both $k$ and $n$. Now verifying the correctness of $M^{(i)}(n)$:

$$|e - f(x)| \leq |e - f(d)| + |f(d) - f(x)|$$

$$\leq 2^{-(n+1)} + |f(d) - f(x)|,$$  

by definition of $\psi$

$$\leq 2^{-(n+1)} + 2^{-(n+1)},$$  

$|d - x| \leq 2^{-m_a(n+1)}$ and definition of $m$

$$= 2^{-n}$$

This completes the first part of the proof. Now assume $f$ is polynomial time computable. We adopt the following notation: for every $x \in \mathbb{R}$ let $\varphi^*_a(n) = \frac{\lceil \log x \rceil}{2}$. Fix some large enough $k$ and consider any $x \in [a,\infty)$ such that $\text{len}([x]) = k + \text{len}([a])$, hence $x \in [a,a+2^k]$. For simplicity in the following discussion we will ignore the constant $\text{len}([a])$. Since $f$ is polytime computable, there exists an oracle Turing machine $M^{(i)}$ such that the computation time of $M^{\varphi_a}(n)$ is bounded by $q(k,n)$ for some polynomial $q$. Fix some large enough $n \in \mathbb{N}$.

Let

$$n_x = \max\{j: \varphi^*_a(j) \text{ is queried during the computation of } M^{\varphi_a}(n+3)\}$$  

(12)

Let $d_x = \varphi^*_a(n_x)$. By the particular choice of Cauchy sequences we have $\varphi^*_a(j) = \varphi^*_a(j)$ for every $j \leq n_x$. Let $\ell_x = d_x - 2^{-n_x}$ and $r_x = d_x + 2^{-n_x}$. Then $\{(\ell_x, r_x): x \in [a,a+2^k]\}$ is an open covering of the compact interval $[a,a+2^k]$. By the Heine-Borel Theorem, $[a,a+2^k]$ has a finite covering $\mathcal{C} = \{(\ell_{x_i}, r_{x_i}): i = 1,\ldots,w\}$. Define $m': \mathbb{N}^2 \rightarrow \mathbb{N}$ by

$$m'(k,n) = \max\{n_{x_i}: i = 1,\ldots,w\}$$  

(13)

First We need to show that $m'$ is a modulus for $f$. Assume some $x,y \in [a,a+2^k]$ such that $x < y$ and $|x-y| \leq 2^{-m'(n)}$.
There exists a function $P_{\text{Time}}$ computable, however, its restriction to $\mathbb{R}$ is not $P_{\text{Time}}$ computable.

Let $P_{\mathbb{R}}[B] = \{ f \in P_{\mathbb{R}}[B] : f \text{ is } P_{\text{Time}} \text{ computable} \}$ and let $P_{\mathbb{D}}[B, \text{cnt, poly}] = \{ f \in P_{\mathbb{D}}[B, \text{cnt}] : f \text{ has a polynomial modulus with respect to both arguments} \}$. Let $P_{\mathbb{R}}[B] \upharpoonright \mathbb{D} = \{ f \in \mathbb{D}[B] : \exists \hat{f} \in P_{\mathbb{R}}[B] \text{ such that } f = \hat{f} \upharpoonright \mathbb{D} \}$. The following result shows the converse of Theorem 15 that there exists a dyadic-preserving real function that is $P_{\text{Time}}$ computable, however, its restriction to $\mathbb{D}$ is not $P_{\text{Time}}$ computable.

Theorem 16. There exists a function $f \in P_{\mathbb{R}}[B] \upharpoonright \mathbb{D}$ that is not $P_{\text{Time}}$ computable.

Proof. Define a function $f : [0, \infty) \to \mathbb{R}$ as follows:

$$f(x) = \begin{cases} 0 & x \in \mathbb{N} \\ \frac{1}{2} + 2^{-2^k} & x = j + \frac{1}{2}, \text{ for } j \in \mathbb{N} \text{ and } k = \min \{ i \in \mathbb{N} : x < 2^i \} \\ 2(x - j)f(j + \frac{1}{2}) & j < x < j + \frac{1}{2}, j \in \mathbb{N} \\ 2(j + 1 - x)f(j + \frac{1}{2}) & j + \frac{1}{2} < x < j + 1, j \in \mathbb{N} \end{cases} \quad (14)$$

$f$ is piecewise linear with breakpoints at $j$’s and $(j + \frac{1}{2})$’s for $j \in \mathbb{N}$. It is zero at the integer points and $\frac{1}{2} + \epsilon_j$ at the midpoints where $\epsilon_j$ is a very small value that depends on the binary length of $j$. The
idea is that real computation is inherently approximate hence to get the exact correct value at \( j + \frac{1}{2} \) the precision input \( n \) has to be large enough (much larger than the extension parameter) making the complexity polynomial in terms of \( n \) although it is exponential in terms of the extension parameter. And therefore the overall complexity is polynomial. On the other hand dyadic computation does not involve this precision parameter leaving the overall computation exponential in terms of the only remaining extension parameter. Now we give the technical details. It is clear that \( f \) preserves \( \mathbb{D} \). Let \( g = f \upharpoonright \mathbb{D} \).

Assume some \( x \in \text{dom}(g) \) such that \( x = j + \frac{1}{2} \) for some \( j \in \mathbb{N} \). Let \( k = \text{len}(j) \). From the definition of \( f \), \( \text{len}(g(x)) = \Omega(2^k) \). Hence \( g \) is not \( \text{PTime} \)-computable as a dyadic function. Now remains to show \( f \) is \( \text{PTime} \)-computable as a real function. Assume some \( x \in \text{dom}(f) \) and assume some \( \varphi \in CF_x \). Let \( M^\varphi \) be an oracle Turing machine such that \( M^\varphi(n) \) does the following:

1. Let \( e = \varphi(2) \),

2. Determine the least \( k \) such that \( e + 1 < 2^k \),

3. Let \( d = \varphi(n+3) \),

4. If \( j \leq d \leq j + \frac{1}{2} \) for some \( j \in \mathbb{N} \), then
   
   a. If \( n \geq 2^k - 10 \), then output \( 2(d - j)(\frac{1}{2} + 2^{-2^k}) \),
   
   b. else output \( (d - j) \),

5. If \( j + \frac{1}{2} \leq d \leq j + 1 \) for some \( j \in \mathbb{N} \), then
   
   a. If \( n \geq 2^k - 10 \), then output \( 2(j + 1 - d)(\frac{1}{2} + 2^{-2^k}) \),
   
   b. else output \( (j + 1 - d) \),

6. End.

Clearly \( M^\varphi(n) \) runs in polynomial time with respect to \( n \) and \( k \). We need to show its correctness. Assume \( \varepsilon = |x - d| \leq 2^{-(n+3)} \). We have the following cases.

**case 1:** \( x, d \in [j, j + \frac{1}{2}] \). If \( n \geq 2^k - 10 \), then

\[
|M^\varphi(n) - f(x)| = |2(d - j)f(j + \frac{1}{2}) - 2(x - j)f(j + \frac{1}{2})| \\
= 2f(j + \frac{1}{2})|x - d| \\
\leq 2(\frac{1}{2} + 2^{-2^k})2^{-(n+3)} \\
= 2^{-(n+3)} + 2^{-(2^k+n+2)} \\
\leq 2^{-(n+2)}
\]
If \( n < 2^k - 10 \), then
\[
|M^p(n) - f(x)| = |(d - j) - 2(x - j)(\frac{1}{2} + 2^{-2^k})| \\
= 2|\frac{1}{2}(d - j) - (x - j)(\frac{1}{2} + 2^{-2^k})| \\
= 2|\frac{1}{2}d - \frac{1}{2}j - \frac{1}{2}x + \frac{1}{2}j - 2^{-2^k}(x - j)| \\
= 2|\frac{1}{2}(d - x) - 2^{-2^k}(x - j)| \\
\leq 2(|\frac{1}{2}(d - x)| + |2^{-2^k}(x - j)|) \\
\leq 2^{-(n+3)} + 2^{-2^k} \\
\leq 2^{-(n+2)}
\]

**case 2:** \( x, d \in [j + \frac{1}{2}, j + 1] \). This case is symmetrical with case 1.

**case 3:** One of \( x \) or \( d \) is in \([j, j + \frac{1}{2}]\) and the other is in \([j + \frac{1}{2}, j + 1]\). Then
\[
|M^p(n) - f(x)| \leq |M^p(n) - f(j + \frac{1}{2})| + |f(j + \frac{1}{2}) - f(x)| \\
\leq 2^{-(n+2)} + 2^{-(n+2)}, \quad \text{from previous cases} \\
= 2^{-(n+1)}
\]

Similar calculations for the case when either \( x \) or \( d \) is in \([j + \frac{1}{2}, j + 1]\) and the other in \([j + 1, j + \frac{3}{2}]\) with \( f(j + \frac{1}{2}) \) replaced by \( f(j + 1) \).

Hence, \( f \) is \( PTime \) computable as a real function and this completes the proof of the lemma. \( \square \)

Theorem 15 and Theorem 16 lead to the following interesting surprising corollary.

**Corollary 17.** There exists \( f \in \mathcal{P}_D[\mathcal{P}, \text{cnt}] \) such that its extension to \( \mathbb{R} \) is not polynomial time computable. And there exists a dyadic-preserving real function \( g \in \mathcal{P}_R[\mathcal{P}] \) such that \( g \upharpoonright \mathbb{D} \) is not polynomial time computable.

This corollary basically states that polynomial time complexity over the reals is not simply an extension of the corresponding notion over the dyadics; this is in spite of the fact that real computation is approximated (in the sense of recursive analysis) by dyadic computation. This can be justified by the following observations: (1) the notion of modulus of continuity does not play any role in the computation of dyadic functions, there even exist efficiently computable dyadic functions that do not have modulus of continuity, so computation of a dyadic function is not related, or at most weakly related, to the smoothness of the function, (2) on the contrary continuity of real functions is a necessary condition for their computability, (3) there are two factors controlling the complexity of computing a dyadic function (and finite objects in general): how hard it is to compute every single bit of the output and the length of the output, and (4) on the other hand there are three factors controlling the complexity of computing a real function (in the sense of recursive analysis); (i) the first, same as in the dyadic case, is how hard it is to compute every single bit of the output, (ii) the second, partially similar to the dyadic case, is the length of the integer part of the output (the length of the fractional part is already controlled by the required precision which is already an input to the machine), and (iii) the third factor (and this is the one absent from the dyadic case) is how hard it is to access the input and this is essentially controlled by the modulus function.
4 Further Research Directions

We intend to pursue the following lines of thought:

1. Investigate polynomial time computability of rational and real functions that are defined over non-compact domains of the form $\mathbb{U}$.
2. Characterize the classes $\mathcal{P}_\mathbb{D}[\mathbb{B},\text{cnt}]$ and $\mathcal{P}_\mathbb{D}[\mathbb{B},\text{disnt}]$ by function algebras. A candidate function algebra (for the continuous case) would be an extension to the dyadics of the Bellantoni and Cook class $\Pi$.
3. Characterize the classes $\mathcal{P}_\mathbb{D}[\mathbb{U},\text{cnt}]$ and $\mathcal{P}_\mathbb{D}[\mathbb{U},\text{disnt}]$ by function algebras.
4. Characterize the classes $\mathcal{P}_\mathbb{R}[\mathbb{B}]$, $\mathcal{P}_\mathbb{R}[\mathbb{U}]$, and $\mathcal{P}_\mathbb{R}$ by function algebras.

References

[1] Stephen Bellantoni & Stephen Cook (1992): A New Recursion-Theoretic Characterization of the Polytime Functions. *Computational Complexity* 2, pp. 97–110.
[2] Olivier Bournez & Emmanuel Hainry (2006): Recursive Analysis Characterized as a Class of Real Recursive Functions. *Fundamenta Informaticae* 74(4), pp. 409–433.
[3] Manuel Campagnolo (2001): Computational Complexity of Real Valued Recursive Functions and Analog Circuits. Ph.D. thesis, Instituto Superior Técnico.
[4] Manuel Campagnolo, Cristopher Moore & José Costa (2000): Iteration, Inequalities, and Differentiability in Analog Computers. *Journal of Complexity* 16(4), pp. 642–660.
[5] Manuel Campagnolo, Cristopher Moore & José Costa (2002): An Analog Characterization of the Grzegorczyk Hierarchy. *Journal of Complexity* 18(4), pp. 977–1000.
[6] Manuel Campagnolo, Cristopher Moore & Joslix Costa (2000): An Analog Characterization of the Subrecursive Functions. In: 4th Conference on Real Numbers and Computers. Odense University Press, pp. 91–109.
[7] A. Grzegorczyk (1955): Computable functionals. *Fundamenta Mathematicae* 42, pp. 168–202.
[8] L. Kalmár (1943): Egyzzerű Példa Eldöntethetlen Aritmetikai Problémára. *Maté és Fizikai Lapok* 50, pp. 1–23.
[9] Ker Ko (1991): Complexity Theory of Real Functions. Birkhäuser.
[10] D. Lacombe (1955): Extension de la Notion de Fonction Récursive aux Fonctions d’une ou Plusieurs Variables Réelles III. *Comptes Rendus de l’Académie des sciences Paris* 241, pp. 151–153.
[11] C. Moore (1996): Recursion Theory on the Reals and Continuous-Time Computation. *Theoretical Computer Science* 162(1), pp. 23–44.
[12] Alan Turing (1936): On Computable Numbers, With an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* 2(42), pp. 230–265. (correction ibid. 43, pp 544-546, 1937).