Electronic Colloquium on Computational Complexity, Report No. 134 (2021)

A Refinement of the Meyer-McCreight Union Theorem

Siddharth Bhaskar

August 19, 2021

Abstract

For a function $t : 2^\star \to 1^\star$, let C_t be the set of problems decidable on input x in time at most $t(x)$ almost everywhere. The *Union Theorem* of Meyer and McCreight asserts that any union $\bigcup_{i<\omega} C_{t_i}$ for a uniformly recursive sequence of bounds t_i is equal to C_L for some single recursive function L. In particular the class PTIME of polynomial-time relations can be expressed as C_L for some total recursive function $L : 2^\star \to 1^\star$. By controlling the complexity of the construction, we show that in fact PTIME $= C_L$ for some L computable in *quasi-polynomial time*.

1 Introduction

Blum [1] observed that several fundamental properties about time complexity classes could be derived abstractly using a few important properties of Turing machines as black boxes; in particular, the property that the relation $\Phi_e(x) \leq n$ is decidable, where $\Phi_e(x)$ is the running time of the Turing machine encoded by e on the input x. (Indeed, the theory applies much more broadly, but we restrict our attention to time on Turing machines.) Using this framework, Meyer and McCreight [4] showed that given any uniformly recursive monotone sequence $t_0 < t_1 < t_2 < \dots$ of time bounds, there exists a recursive function t such that a problem is decidable in time t iff it is decidable in time t_i for some i. If C_t is the set of problems decidable in time t almost everywhere, we have $C_t = \bigcup_{i<\omega} C_{t_i}$. This result is known as the *Union Theorem*.

Consequentially, there is a single recursive function t such that the class C_t is equal to PTIME, and the same holds for any complexity class which can be naturally expressed as a length-ω union of DTIME$(t_i(n))$ for a monotone sequence of recursive functions t_i. The time bound t is super-polynomial, but only just.

In this note, we show that PTIME is not only equal to C_t for some recursive function t, but in fact that we can locate such a t within the class of *quasi-polynomial time* functions, i.e., those computable in time $2^{(\log n)^{O(1)}}$. The starting point of our investigation is that the "Blum relation" $\Phi_e(x) \leq n$ is not only decidable, but is decidable quite efficiently, in particular within PTIME. Since the Meyer-McCreight construction of t from t_i does not use any unbounded minimization, we can estimate the complexity of t from the complexity of t_i and the complexity of the Blum relation. The main technical innovation of our paper is a "delayed" version of the original diagonalization, plus an analysis of its complexity.

Our paper is structured as follows: in Section 2, we review the fundamental background information and notation; in Section 3, we prove the Blum relation is PTIME; in Section 4, we present the delayed priority construction; and in Section 5, the complete theorem. Finally, in Section 6, we discuss the significance and extensions of our work.

ISSN 1433-8092

2 Preliminaries

Let ω be the set of natural numbers and $\{0,1\}^\star$ be the set of finite binary strings. We adopt the convention, common in set theory, that identifies a natural number n with its set of predecessors $\{0,1,\ldots,n-1\}$. In particular, $2 = \{0,1\}$, and $2^\star$ is the set of binary strings. Similarly, $1^\star$ is the set of unary strings. Given a string x, $|x| \in \omega$ is its length. We fix a polynomial-time bijection $\langle\cdot,\cdot\rangle : 2^\star \times 2^\star \to 2^\star$ with inverse functions π_0 and π_1. (Colloquially, these are "pairing" and "unpairing" functions respectively.)

In this paper, we shall have to deal with two different encodings of natural numbers by strings, viz., by unary and binary numerals. To that end, we fix two bijections $1^\star \simeq \omega$ and $2^\star \simeq \omega$, and treat these as two separate copies of the natural numbers. The bijection $1^\star \simeq \omega$ is simply given by $x \mapsto |x|$. We identify $\{1,2\}^\star$ with ω using the bijection

$$b_0 b_1 \ldots b_{n-1} \mapsto b_0 + 2b_1 + \cdots + 2^{n-1} b_{n-1};$$

then by fixing a bijection of $\{1,2\}$ with 2, we get an identification of $2^\star \simeq \omega$. Under this identification, the length of a number is proportional to the logarithm of its magnitude, and that the arithmetic operations $+, \times, -, \div$ as well as integer comparison $<$ are all computable in polynomial time.

For any sets X and Y, let $X \rightharpoonup Y$ be the set of partial functions from X to Y. If $f : X \rightharpoonup Y$, then by $f(x) = y$, $f(x) \leq y$, etc., we mean in particular that x is in the domain of convergence of f. Let $\exists^\infty$ and $\forall^\infty$ mean "there exist infinitely many" and "for all but finitely many" respectively. If $P(x)$ is a unary predicate on the natural numbers, then by $(\mu x)P(x)$ we mean the least natural number satisfying P, if it exists. (We can also form $(\mu x)P(x)$ for $P \subseteq 1^\star$ or $P \subseteq 2^\star$ via the above identifications of these sets with ω.)

We assume familiarity with the basics of Turing machines and time complexity classes. For a function $f : \omega \to \omega$, $\textsc{Dtime}(f(n))$ is the class of all relations decidable by a deterministic Turing machine in time at most $f(n)$ for inputs of total length n. (By a *relation* we mean a subset of $(2^\star)^n$ for some n; thus complexity classes are classes of relations, not just languages, i.e., monadic relations. The *total length* is the sum of the lengths of the inputs.) We also assume familiarity with fundamental complexity classes like Ptime, Pspace, and Exptime. We define the class of *quasi-polynomial time relations* by

$$\textsc{QPtime} = \bigcup_{c<\omega} \textsc{Dtime}(2^{(\log n)^c}).$$

Notice that $\textsc{Ptime} \subsetneq \textsc{QPtime} \subsetneq \textsc{Etime}$, where $\textsc{Etime} = \bigcup_{c<\omega} \textsc{Dtime}(2^{cn})$.

For a given relational class, we will typically prefix it by "F" to indicate the corresponding class of functions. For example, FPtime is the class of functions computable by a Turing machine in polynomial time. Note that we are being deliberately ambiguous about the *type* of these functions; they could be, for example $2^\star \to 2^\star$ or $2^\star \to 1^\star$. We identify each relation with its characteristic function; this gives us a containment of each relational class in the corresponding functional class.

We collect several important properties of QPtime into a lemma. But first, a definition.

Definition 1. A *quasi-polynomial time bound* is a function $q : \omega^n \to \omega$ of the form $2^{p(\log a_1,\ldots,\log a_n)}$ for some polynomial $p : \omega^n \to \omega$ defined by an element of $\mathbb{N}[x_1,\ldots,x_n]$.

Lemma 1. *The following properties hold of* QPtime *and* FQPtime.

1. A relation $R(x_1, \ldots, x_n)$ is in QPTIME iff there is a quasi-polynomial time bound $q : \omega^n \to \omega$ such that R is decidable by some Turing machine in time $q(|x_1|, \ldots, |x_n|)$. f

2. The function $f : 2^\star \times 2^\star \to 1^\star$ defined by $f(x, y) = |x|^{\log |y|}$ is in FQPTIME.

3. Quasi-polynomial time bounds are closed under composition.

4. QPTIME relations are closed under FQPTIME many-one reductions; i.e., if $R \in$ QPTIME and $f \in$ FQPTIME, then $R(f(x)) \in$ QPTIME.

5. QPTIME is self-low; i.e., any relation computable in quasi-polynomial time with respect to a quasi-polynomial time oracle is again in quasi-polynomial time; i.e., $\text{QPTIME}^{\text{QPTIME}} =$ QPTIME.

Proof. We briefly sketch the proof of each item:

1. We have to show that each function of the form $2^{(\log(a_1+\cdots+a_n))^c}$ is bounded by a quasi-polynomial time bound in $(a_1, \ldots, a_n)$ and vice-versa. For the first direction, by the AM-GM inequality, for any $n \geq 1$ and natural numbers $a_1, \ldots, a_n$,

$$a_1 + \cdots + a_n \leq n(a_1 \ldots a_n)^{\frac{1}{n}} \leq n a_1 \ldots a_n.$$

   Hence by applying the monotone function $x \mapsto 2^{(\log x)^c}$ to the left and right sides,

$$2^{(\log(a_1+\cdots+a_n))^c} \leq 2^{(\log n+(\log a_1+\cdots+\log a_n))^c}.$$

   In the other direction, given a quasi-polynomial time bound $q(a_1, \ldots, a_n)$ we have that

$$q(a_1, \ldots, a_n) \leq q(a_1 + \cdots + a_n, \ldots, a_1 + \cdots + a_n).$$

   The latter term has the form $2^{p(\log(a_1+\cdots+a_n))}$ for some polynomial p; bound p by some monomial term.

2. Given y, we can calculate the binary numeral encoding $|y|$ in polynomial time, and then the binary numeral encoding $\log |y|$ again in polynomial time. Let us treat x as a unary numeral. Now the computational task is to raise a unary numeral to a binary exponent, and the time required by that task is no more than the space required to write the output, which is $|x|^{\log |y|} = 2^{\log |x| \log |y|}$, visibly a quasi-polynomial time bound.

3. This follows from the closure of natural-number polynomials under composition.

4. If R is decidable in time q_1 and f is computable in time q_2, both quasi-polynomial time bounds, then $R \circ f$ is computable in time $q_2 + q_1 \circ q_2$, which is dominated by a quasi-polynomial time bound.

5. If P is decidable in time q_1 relative to an oracle R itself decidable within a quasi-polynomial time bound q_2, then P is decidable absolutely in time $q_1 \cdot q_2 \circ q_1$, which is dominated by a quasi-polynomial time bound.

□

A note of caution. Complexity classes are parameterized by time bounds, which are functions of type $2^\star \to \omega$. (Given a time bound t, we can form the class C_t of all relations decidable within time t almost everywhere.) When we consider the complexity of computing these times bounds, it is important to distinguish whether by ω we mean $1^\star$ or $2^\star$.

Consider, for example, the function $t(x) = 2^{|x|^2}$. As a function $2^\star \to 1^\star$ this is certainly not polynomial-time computable—the output is too long—but, as a function $2^\star \to 2^\star$, it *is*. In the latter case, we are in the strange (though not contradictory!) situation of having a PTIME function t which defines complexity class containing ETIME.

On the other hand, if we restrict our time bounds to functions of type $2^\star \to 1^\star$, then PTIME enjoys the *Ritchie-Cobham property*; viz., PTIME is the union of all C_t for t computable in polynomial time (cf. Odifreddi [5]). In other words, PTIME is exactly the set of all relations which can be computed within some polynomially-computable running time. Moreover, no such C_t exhausts all of PTIME by the Time Hierarchy theorem.

In this paper, we exhibit a quasi-polynomial time computable function t of type $2^\star \to 1^\star$ such that $C_t = \text{PTIME}$. This is stronger than exhibiting such a function of type $2^\star \to 2^\star$. Moreover, there is a good corresponding lower bound: we know that such a t cannot be computable in polynomial time.

3 A Blum structure

Let us fix a "standard" encoding of Turing machines by binary strings. We will not specify the encoding, but rather state the crucial property that we need it to satisfy, namely that it admits an *efficient universal machine*. Efficient universal machines are required to justify the time and space hierarchy theorems; every encoding of Turing machines presented in a textbook or classroom admits them.

Time-efficient universal machines were studied by Hartmanis and Stearns [2] and refined by Hennie and Stearns [3]. The following theorem is a form of the main result of the latter paper but does not appear verbatim in either. In fact, it is not a theorem at all, since the phrase *simulate for n steps* has no formal meaning. However, we regard this, and related informal statements concerning the operation of Turing machines, as part of the "folk knowledge" of the complexity theory community. In particular, we believe that we all understand the same thing by this, and that it requires no further explanation.

Theorem 1 (Existence of time-efficient universal Turing machines)**.** *There is an encoding of Turing machines M_α by strings α and a universal Turing machine U, such that for every string α encoding a machine and string $x \in 2^\star$, $U(\alpha, x) = M_\alpha(x)$ (in particular, one side diverges iff the other one does). Moreover,*

1. *it is decidable in polynomial time whether a string encodes a machine,*
2. *for every α encoding a machine, U simulates n steps of the computation of M_α on input x within $O(|\alpha| n \log n)$ steps.*

Definition 2. For strings $\alpha \in 2^\star$ which encode machines, let $\Phi_\alpha : 2^\star \rightharpoonup 1^\star$ be the running time of M_α.

The notation Φ_α is supposed to recall *Blum structures*, which are an indexing φ_e of partial recursive functions, along with a "resource bound" Φ_α for each program code e, such that the relation $\Phi_\alpha(n) < m$ is recursive.

Now we state two fundamental lemmas about the complexity of computing the running time of a given machine on given inputs. The first of these is the refinement of the classical Blum property above. That is to say, for an encoding of Turing machines with an efficient universal function, not only is $\Phi_\alpha(x) < m$ computable, but it's efficiently computable as well.

Note that this is the central gambit of this paper—if the Blum relation is not only computable, but within some complexity class, can we control the complexity of other constructions made using Blum structures?

Lemma 2. *The relation $P(\alpha, x, y)$ defined by*

$$(\alpha \textit{ encodes a machine }) \wedge (\Phi_\alpha(x) < |y|)$$

is contained in PTIME.

Proof. Given α, x, and y, first test whether α encodes a Turing machine, in time $|\alpha|^{O(1)}$. If so, using y as a counter, run the universal machine U on inputs (α, x) for $O(|\alpha||y| \log |y|)$ steps, to simulate $M_\alpha(x)$ for $|y|$ steps. Output true or false depending on whether or nor M_α converges within this time. The complexity of the whole process is $(|\alpha||y|)^{O(1)}$. □

The next lemma concerns the complexity of computing *partial* functions. For a partial function f to be computed within time t means that there is a Turing machine computing f whose running time is bounded by t *on convergent inputs.*

Lemma 3. *If $g : 2^\star \to 1^\star$ is in* FQPTIME*, then the partial function*

$$f(\alpha, x) = \begin{cases} \Phi_\alpha(x) & \textit{if } \Phi_\alpha(x) < g(x) \\ \uparrow & \textit{otherwise} \end{cases}$$

is computable in quasi-polynomial time.

Proof. On inputs α and x, first compute $n = g(x)$ in unary, in quasi-polynomial time. Using n as a counter, run the universal machine $U(\alpha, x)$ for $O(|\alpha| n \log n)$ steps, to simulate $M_\alpha(x)$ for n steps. Since n is bounded by $2^{|x|^c}$ for some c, $O(|\alpha| n \log n)$ is quasi-polynomial time relative to x and α. □

4 Efficient quasi-selection and enumeration

In recursion theory, there are all sorts of constructions along the general lines of: given a relation $R(x, y)$, construct a (total or partial) function f with certain properties. For example, a *selection function* f satisfies $R(x, f(x))$ if there exists a y such that $R(x, y)$, and an *enumeration function* enumerates the image of R. Furthermore, if R is computable, then f should be too.

In this section, given an arbitrary relation R, we want to construct a partial function f which has some properties of both a selection and enumeration function. The problem is, these properties are mutually contradictory: for example, we might imagine that the graph of R is a vertical line, i.e., $(\exists! x)(\exists y)R(x, y)$, but for that unique x-value x_0, $(\forall y)R(x_0, y)$. In this case, the domain of convergence of any selection function is finite, but the domain of convergence of any enumeration function is infinite.

However, we can make these properties consistent by weakening them. For selection, we only require that $R(x, f(x))$ when $f(x)$ converges. For enumeration, we only require that the image of f contains the *infinitely-often image* $\{y : (\exists^\infty x) R(x, y)\}$. (Notice that "weak selection" can be satisfied in and of itself by the always-diverging function, but it becomes nontrivial in the presence of "weak enumeration," which forces convergence at certain values.) Moreover—importantly—we will show that the complexity of f can be controlled by the complexity of R.

The definition follows. Instead of literally defining a partial function f from the relation R, we shall define and work with a binary relation G, but this is simply the graph of f.

Definition 3. For a given relation $R \subseteq \omega \times \omega$, define the relation $G \subseteq \omega \times \omega$ by recursion on x as follows:

$$G(x, y) \iff y = (\mu y' < \log x)\big[R(x, y') \wedge (\forall x' < \log x)\, \neg G(x', y)\big].$$

If there is no such y', then $\neg G(x, y)$ for all y. (In particular, this yields the base case $\neg G(0, y)$.)

Lemma 4. *Let G be obtained from R as in Definition 3. Then*

- *for each y, its G-preimage $\{x : G(x, y)\}$ is finite,*
- *for each x, its G-forward image $\{y : G(x, y)\}$ is finite,*
- *if there is a y such that $G(x, y)$, then there is a $y < \log x$ such that $G(x, y)$,*
- *G is contained in R, and*
- *the image $\{y : (\exists x)\, G(x, y)\}$ of G contains the "infinitely-often image" $\{y : (\exists^\infty x)\, R(x, y)\}$ of R.*

Proof. The first four bullet points are straightforward. Notice that G is the graph of a partial function: for each x, if $G(x, y)$ for some y, then $G(x, y)$ for a unique y, and moreover that y is less than $\log x$. Hence for each x, its G-forward image $\{y : G(x, y)\}$ is finite (in fact of size 0 or 1). If $G(x, y)$ for *any* y, then $y < \log x$.

For each y, if there exists an x such that $G(x, y)$, then for every $x' > 2^x$, the definition of G ensures that $\neg G(x' + 1, y)$. Hence for each y, the inverse image $\{x : G(x, y)\}$ is finite. Finally, if $G(x, y)$, then $R(x, y)$, again by definition of G.

Let us consider the final bullet point. Fix some y_0, and suppose that $R(x, y_0)$ for infinitely many x; let X be the set of such x. Suppose by contradiction that for all x, $\neg G(x, y_0)$. Then

$$(\forall x \in X)\big[R(x, y_0) \wedge (\forall x' < \log x)\, \neg G(x', y_0)\big].$$

However

$$(\forall x \in X)(\exists y < y_0)\big[R(x, y) \wedge (\forall x' < \log x)\, \neg G(x', y)\big],$$

for, if y_0 were the *least* witness y for some $x_0 \in X$, then $G(x_0, y_0)$ by definition of G, contradicting the assumption that $(\forall x)\, \neg G(x, y_0)$.

Abbreviate $R(x, y) \wedge (\forall x' < \log x)\, \neg G(x', y)$ by $Q(x, y)$. Then we have $(\forall x \in X)\, Q(x, y_0)$ but $(\forall x \in X)(\exists y < y_0)\, Q(x, y)$. Since X is an infinite set and $\{y : y < y_0\}$ is finite, we may conclude $(\exists y < y_0)(\exists^\infty x \in X)\, Q(x, y)$.

Let y_1 be the least witness $y < y_0$ to $(\exists^\infty x \in X)\, Q(x, y)$. Then $(\forall y < y_1)(\forall^\infty x \in X)\, \neg Q(x, y)$. Exchanging quantifiers, $(\forall^\infty x \in X)(\forall y < y_1)\, \neg Q(x, y)$. Since $(\exists^\infty x \in X)\, Q(x, y_1)$, we may pick a witness x_0 satisfying $x_0 > 2^{y_1}$, $Q(x_0, y_1)$, and $(\forall y < y_1)\, \neg Q(x_0, y)$. Then

$$y_1 = (\mu y < \log x_0)(Q(x_0, y)),$$

and hence $G(x_0, y_1)$ by definition of G.

However, since $(\exists^\infty x \in X)\, Q(x, y_1)$, we may pick $x_1 \in X$ such that $Q(x_1, y_1)$ and $x_1 > 2^{x_0}$. By definition of Q, $(\forall x' < \log x_1)\, \neg G(x', y_1)$, but this contradicts $G(x_0, y_1)$. □

A game for G. Suppose that $y < \log x$ and $R(x, y)$. Then there are two ways in which $G(x, y)$ could fail. Either there could be a strictly smaller $y' < y$ such that $G(x, y')$, or there could be some $x' < \log x$ such that $G(x', y)$. This suggests the following game for G:

Definition 4. Define a game $\mathcal{G}$ as follows. The *positions* of $\mathcal{G}$ are all pairs $(x, y) \in \omega \times \omega$ such that $y < \log x$ and $R(x, y)$. Given a position (x, y) the valid *moves* consist of all positions of the form (x, y') such that $y' < y$, plus all positions of the form (x', y) such that $x' < \log x$.

There are two players, **I** and **II**. Player **I** moves first from a given starting position (x, y) and players alternate thereafter. A *play* is a sequence of positions $((x_0, y_0), (x_1, y_1), \dots)$ such that each $(x_i, y_i) \to (x_{i+1}, y_{i+1})$ is a valid move. Notice that each play of $\mathcal{G}$ must be finite. A play $((x_0, y_0), \dots, (x_n, y_n))$ is *terminal* in case there are no valid moves from (x_n, y_n). A terminal play is *winning for* **I** in case n is odd, otherwise it is *winning for* **II**. (In other words, if some player has no permissible moves, the other player wins.)

Finally, let $\mathcal{G}^\dagger$ be the "dual game" in which player **II** moves first, and let us denote by, e.g., $\mathcal{G}(x, y)$ the game $\mathcal{G}$ from the initial position (x, y).

Remark 1. Notice that for any (x, y), $\mathcal{G}(x, y)$ is *determined*, i.e., either **I** has a winning strategy or **II** does. Furthermore, **I** has a winning strategy in $\mathcal{G}(x, y)$ iff **II** has a winning strategy in $\mathcal{G}^\dagger(x, y)$.

Finally, $\mathcal{G}$ and $\mathcal{G}^\dagger$ can be defined by simultaneous recursion as follows. Given a position (x, y) of $\mathcal{G}$, player **I** moves to a position (x', y') of $\mathcal{G}^\dagger$. Similarly, given a position (x, y) of $\mathcal{G}^\dagger$, player **II** moves to a position (x', y') of $\mathcal{G}$.

The point is that G can be characterized by who wins $\mathcal{G}$:

Lemma 5. *Suppose $y < \log x$ and $R(x, y)$. Then $G(x, y)$ iff player* **II** *has a winning strategy in the game $\mathcal{G}(x, y)$.*

Proof. By induction on $x + y$. Suppose $G(x, y)$. Then for every $y' < y$, it must be the case that $\neg G(x, y')$, so player **I** has a winning strategy in $\mathcal{G}(x, y')$ by induction, and hence player **II** has a winning strategy in $\mathcal{G}^\dagger(x, y')$. Similarly, for every $x' < \log x$, it must be the case that $\neg G(x', y)$, so player **II** has a winning strategy in $\mathcal{G}^\dagger(x', y)$.

Therefore, if $G(x, y)$, then no matter which valid move player **I** makes from $\mathcal{G}(x, y)$, player **II** has a winning strategy in the resulting instance of $\mathcal{G}^\dagger$. Hence player **II** has a winning strategy in $\mathcal{G}(x, y)$.

Conversely, suppose that $\neg G(x, y)$. Then there is either some $y' < y$ such that $G(x, y')$ or some $x' < \log x$ such that $G(x', y)$. By induction, this means that there is either some $y' < y$ such that player **I** has a winning strategy in $\mathcal{G}^\dagger(x, y')$ or some $x' < \log x$ such that player **I** has a winning strategy in $\mathcal{G}^\dagger(x', y)$. Therefore, player **I** has a winning strategy in $\mathcal{G}(x, y)$, by choosing to move to the appropriate position of $\mathcal{G}^\dagger$. □

Corollary 1. *Suppose $((x_0, y_0), \dots, (x_n, y_n))$ is a play of $\mathcal{G}$ in which both players move optimally. Then for each $0 \leq i < n$, $G(x_i, y_i) \iff \neg G(x_{i+1}, y_{i+1})$.*

Proof. The proof is by induction on n; there is nothing to prove if $n = 0$. Suppose that $G(x_0, y_0)$, so that player **II** has a winning strategy in $\mathcal{G}(x_0, y_0)$. Then player **II** has a winning strategy

in $\mathcal{G}^\dagger(x_1, y_1)$, so by Lemma 5, $\neg G(x_1, y_1)$, and by induction, $G(x_i, y_i) \iff \neg G(x_{i+1}, y_{i+1})$ for $1 \leq i < n$. Since $G(x_0, y_0)$, we get $G(x_i, y_i) \iff \neg G(x_{i+1}, y_{i+1})$ for $0 \leq i < n$. The proof is similar (switching the roles of **I** and **II**) if $\neg G(x_0, y_0)$. □

Now we tackle the question of *deciding* G efficiently.

Definition 5. For any position (x, y) *game tree* of $\mathcal{G}(x, y)$ is the set of all valid plays starting with (x, y). (This is a tree in the sense that it's a prefix-closed set of sequences.)

To evaluate $G(x, y)$, the idea is to construct the game tree for $\mathcal{G}(x, y)$, and then evaluate it using the so-called "minimax" algorithm. This performs a depth-first traversal of the game tree, recursively evaluating the winning positions at all children of (x, y), and using that to evaluate who wins at (x, y). This algorithm takes time polynomial in the size of the game tree, so it suffices to analyze the complexity of constructing the game tree.

The problem is that the game tree of $\mathcal{G}(x, y)$ might be too large relative to $|x|$. (We only care about (x, y) for which $y < \log x$, so we may take $|x|$ to be the only parameter.) The reason is that from any position there may be plays of length $\log x \approx |x|$. (Consider (x, y) where $y = \log x$, and R is always true. Then $((x, y), (x, y-1), (x, y-2), \dots)$ is a play of length $\log x$.) This means that the size of the game tree may be exponential in $|x|$ in general—too big to construct in quasipolynomial time.

Our way around this is to prune some strategies where one or the other player does not make his or her best move. This is a common trick in game-playing algorithms: to evaluate which player wins from a given position, we can ignore moves which we know to be sub-optimal, and thus reduce the size of the tree we need to search through. Indeed, we will show that any play that follows a winning strategy must be very short relative to the length of x. The resulting subtree of the game tree we obtain will be small enough to construct in quasi-polynomial time.

Definition 6. Let a move in $\mathcal{G}$ of the form $(x, y) \mapsto (x', y)$ for $x' < \log x$ be a move of type (a), and a move of the form $(x, y) \mapsto (x, y')$ for $y' < y$ be a move of type (b).

Lemma 6. *Suppose $((x_0, y_0), \dots, (x_n, y_n))$ is a play of $\mathcal{G}$ in which both players play optimally. Then there cannot be 3 (or more) consecutive moves of type (b).*

Proof. Suppose that $(x, y) \mapsto (x, y') \mapsto (x, y'') \mapsto (x, y''')$ were 3 consecutive moves of type (b); then $y > y' > y'' > y'''$. By Corollary 1, either $G(x, y)$ and $G(x, y'')$ or $G(x, y')$ and $G(x, y''')$. But either case is contradictory, since G is the graph of a partial function. □

Remark 2. A play starting with (x, y) in which no three moves of type (b) are made consecutively is very short relative to the magnitude of x. Namely, at least every third move, the x-value must decrease by at least a logarithm. Hence the length of any such play is bounded above by $3 \log^\star x$, where $\log^\star$ is the iterated logarithm. The iterated logarithm $\log^\star x$ certainly grows slower than, say, $\log |x| \approx \log(\log x)$.

Theorem 2. *If $R \in$ QPTIME, then $G \in$ QPTIME.*

Proof. We present a quasi-polynomial time decision procedure for G. Given (x, y), first verify that $y < \log x$ and $R(x, y)$, otherwise reject. Construct the game tree of $\mathcal{G}(x, y)$, omitting any plays which have three or more consecutive moves of type (b).

From a position (x, y), there are most $\log x + y \leq 2 \log x$ valid moves, which means that every node has at most $2 \log x = 2|x|$ children. Moreover, the depth of this tree is at most $3 \log |x|$, as

observed above. Hence, the size of this tree is bounded by $(2|x|)^{3\log|x|} = 2^{O((\log|x|)^2)}$, a quasi-polynomial in $|x|$. The time required to construct this tree is bounded by a polynomial in the size of the tree, so itself is a quasi-polynomial in $|x|$.

Finally, use the minimax algorithm applied to this game tree to determine the winner. The complexity of this algorithm is again bounded by a polynomial in the size of the tree, which again is a quasi-polynomial in $|x|$. □

5 The union theorem

In this section, we prove a refinement of the Meyer-McCreight Union Theorem. The construction of L is essentially identical to the construction in the original proof; all we must do is estimate its complexity using the results of the previous section.

An important definition in this section is that of a *complexity class* C_f given a function f.

Definition 7. If $f : 2^\star \to 1^\star$, then C_f is the class of decision problems decidable in time f almost everywhere, i.e.,

$$C_f = \{X \subseteq 2^\star : (\exists \alpha)\, M_\alpha \text{ decides } X \wedge (\forall^\infty x)\, \Phi_\alpha < f(x)\}.$$

Definition 8. Define $t : 2^\star \times 2^\star \to 1^\star$ by $t(x,y) = |x|^{\log|y|}$; define $t_y(x) = t(x,y)$.

As observed previously, $t \in$ FQPTIME. When writing t_y, we will typically think of y as living in ω, using the bijection $\omega \simeq 2^\star$ specified earlier. Notice that $C_{t_0} \subseteq C_{t_1} \subseteq \ldots$ and the union of this sequence is PTIME. The objective in this section is to construct an $L : 2^\star \to 1^\star$ in FQPTIME such that $C_L = \bigcup_{y<\omega} C_{t_y} =$ PTIME.

Definition 9. Define $R \subseteq 2^\star \times 2^\star$ by

$$R(x,y) \iff t_y(x) < \Phi_{\pi_0(y)}(x) \leq t_x(x).$$

Remark 3. Notice that R is obtained by taking the polynomial-time relation $Q \subseteq 1^\star \times 2^\star \times 2^\star \times 1^\star$ given by

$$Q(u,e,w,z) \iff u < \Phi_\alpha(w) \leq z$$

and substituting $u \leftarrow t_y(x)$, $e \leftarrow \pi_0(y)$, $w \leftarrow x$, and $z \leftarrow t_x(x)$. Since these are all FQPTIME-computable functions, $R \in$ QPTIME by the closure of QPTIME under FQPTIME reductions.

Definition 10. Define $G \subseteq 2^\star \times 2^\star$ from R as in Lemma 3, viz.,

$$G(x,y) \iff y = (\mu y' < \log x)\big[R(x,y') \wedge (\forall x' < \log x)\, \neg G(x',y)\big].$$

If there is no such y', then $\neg G(x,y)$ for all y.

Since $R \in$ QPTIME, $G \in$ QPTIME by Theorem 2.

Definition 11. Define $L : 2^\star \to 1^\star$ by

$$L(x) = \begin{cases} \Phi_{\pi_0(y_\star)}(x) - 1 & \text{if } y_\star = (\mu y)\, G(x,y) \\ t_x(x) & \text{if } \neg(\exists y)\, G(x,y). \end{cases}$$

Remark 4. Observe that $(\forall x \in 2^\star)\, L(x) \leq t_x(x)$.

Lemma 7. $L \in$ FQPTIME.

Proof. Consider first the problem of, given x, finding out whether there exists a $y \in 2^\star$ such that $G(x, y)$, and if so, finding the least one $y_\star$. By Lemma 4, we may restrict our search to $y < \log x$, which is $|x|$ many strings. Since testing $G(x, y)$ for each $y < \log x$ takes time bounded by a fixed quasi-polynomial bound in $|x|$, this whole procedure runs in quasi-polynomial time.

If there exists a least witness $y_\star$ to $G(x, y)$, then in particular $R(x, y_\star)$, so $\Phi_{\pi_0(y_\star)}(x) \leq t_x(x)$. Since $x \mapsto t_x(x) : 2^\star \to 1^\star$ is contained in FQPTIME, the partial function

$$f(\alpha, x) = \begin{cases} \Phi_\alpha(x) & \text{if } \Phi_\alpha(x) \leq t_x(x) \\ \uparrow & \text{otherwise} \end{cases}$$

is computable in FQPTIME. Hence, so is the function $f(\pi_0(y_\star), x)$. But this is exactly the function we need to evaluate in this case.

Otherwise, if there is no witness y to $G(x, y)$, we simply evaluate and return $t_x(x)$, in quasi-polynomial time. □

Theorem 3. $C_L = \bigcup_{y<\omega} C_{t_y}$.

Proof. We must show two containments, $\subseteq$ and $\supseteq$. For the latter, it suffices to show that for each $z < \omega$, $C_{t_z} \subseteq C_L$. To show this in turn, it suffices to show that L dominates each t_z cofinitely often; i.e., that for each z, there are at most finitely many solutions x to $L(x) < t_z(x)$. Fix z. It suffices to partition $2^\star$ into finitely many pieces and show that $L(x) < t_z(x)$ has at most finitely many solutions from each piece.

Consider x such that $G(x, y)$ has no solutions in y. For such x, $L(x) = t_x(x)$, and for any solution to $L(x) < t_z(x)$, $|x|^{\log|x|} < |x|^{\log|z|}$. For sufficiently long x the left-hand side is larger; therefore, there are at most finitely many solutions x to $|x|^{\log|x|} < |x|^{\log|z|}$.

On the other hand, consider x such that $G(x, y)$ for some y. Let $y_\star = y_\star(x)$ be the least witness to $G(x, y)$; for such x, $L(x) = \Phi_{\pi_0(y_\star)}(x) - 1$. For each x, $G(x, y_\star)$ implies $R(x, y_\star)$; in particular, $t_{y_\star}(x) < \Phi_{\pi_0(y_\star)}(x)$, and hence $t_{y_\star}(x) \leq L(x)$. Therefore any solutions to $L(x) < t_z(x)$ must satisfy $t_{y_\star}(x) < t_z(x)$, and thus $|y_\star| \leq |z|$.

In other words, for each such x, $(\exists y)\, |y| \leq |z| \wedge G(x, y)$. But the set of such x can be expressed as a finite union (over all y such that $|y| \leq |z|$) of the G-pre-image of y, each of which are finite by Lemma 4. This concludes the proof of the containment $\supseteq$.

It remains to show that $C_L \subseteq \bigcup_{n<\omega} C_{t_n}$. We shall show that for every program code α, if M_α decides a language outside $\bigcup_{n<\omega} C_{t_n}$, then it is not in C_L either. Fix such an α. The hypothesis entails

$$(\forall z < \omega)(\exists^\infty x)\, t_z(x) < \Phi_\alpha(x).$$

Suppose by contradiction that M_α decides a language in C_L, i.e., $(\forall^\infty x)\, \Phi_\alpha(x) < L(x)$. Then in particular,

$$(\forall z < \omega)(\exists^\infty x)\, t_z(x) < \Phi_\alpha(x) \leq L(x).$$

Since $L(x) \leq t_x(x)$ for all x,

$$(\forall z < \omega)(\exists^\infty x)\, t_z(x) < \Phi_\alpha(x) \leq t_x(x).$$

Substituting $\langle \alpha, z \rangle$ for z,

$$(\forall z < \omega)(\exists^\infty x)\, t_{\langle \alpha, z \rangle}(x) < \Phi_\alpha(x) \leq t_x(x),$$

i.e., $(\forall z < \omega)(\exists^\infty x)\, R(x, \langle \alpha, z\rangle)$. By Lemma 4, the infinitely-often image of R is contained in the image of G, so $(\forall z < \omega)(\exists x)\, G(x, \langle \alpha, z\rangle)$. Let $Z = \{y : (\exists z)\, y = \langle \alpha, z\rangle\}$ be the set of all y such that $\pi_0(y) = \alpha$. Then Z is an infinite set, since $z \mapsto \langle \alpha, z\rangle$ is an injection, and Z is contained in the image of G.

By Lemma 4, the forward image of any finite set is finite. Therefore, the preimage of Z must be infinite. But for those infinitely many x in the G-preimage of Z, $L(x) \leq \Phi_\alpha(x)$. This is because for such x, $G(x, y)$ for a unique $y \in Z$; for this y, $t_y(x) < \Phi_\alpha(x)$, but also $L(x) = \Phi_\alpha(x) - 1$. This contradicts the assumption that $\Phi_\alpha(x) < L(x)$ for cofinitely many x.

Hence, the language decided by M_α is not contained in C_L. Since α was arbitrary, $C_L \subseteq \bigcup_{z<\omega} C_{t_z}$, which is what we needed to show.

□

6 Discussion

We have obtained a dramatic reduction in the known complexity of any time bound defining PTIME. The original statement of the Union Theorem only guarantees that the time bound is recursive; perhaps a back-of-the-envelope calculations shows that it can be computable in exponential space. On the other hand, we have located such a bound within FQPTIME.

A natural question to ask is, "how general is our argument?" On one hand, it seems hard to mirror it in any smaller class. We use, seemingly crucially, the fact that polynomial time bounds are closed under composition, as well as the fact that the binary logarithm is computable in polynomial time. For larger complexity classes with similar closure properties (like elementary-time or primitive recursive relations), it is not clear what the corresponding analogue of QPTIME is. However, we do find it plausible that our results relativize, i.e., for any (relational) oracle O, $\text{PTIME}^O = C_L^O$ for some $L \in \text{FQPTIME}^O$. If true, this says something to the effect that this phenomenon holds "in a neighborhood" of PTIME.

As to the utility of our result, we believe that PTIME is of such fundamental interest that anything novel and non-obvious we can say about it is worth knowing. However, a natural question suggested by our work is, for a given complexity class C, what the minimal time or space bound t is that we need to express C as C_t. Odifreddi [5] notes that *practically every natural class of recursive functions which is sufficiently rich is a complexity class for measures such as time or space complexity.*

One might expect that for "most" classes C, the complexity of any t such that $C = C_t$ might be large relative to the complexity of decision problems contained in C. If true, this might imply that PTIME is special in some sense: the t in question is not that complex compared to elements of PTIME.

Finally, we raise the question of what other constructions in Blum's theory might be mined for complexity-theoretic information. Certain constructions contain an unbounded search, and may not admit a complexity-theoretic refinement, no matter how efficient the Blum relation. However, as we have shown here, some constructions may be made very efficient. Looking for others strikes us as an important endeavor, with plausibly new consequences.

References

[1] M. Blum: A Machine-Independent Theory of the Complexity of Recursive Functions. *Journal of the ACM* 14 (1967) 322-336.

[2] J. Hartmanis and R.E. Stearns: On the Computational Complexity of Algorithms, *Trans. Am. Math. Soc.* 117 (1965) 285–306.

[3] F. C. Hennie and R. E. Stearns: Two tape simulation of multitape Turing machines. *Journal of the ACM* 13 (1966) 533-546.

[4] A. R. Meyer and E. M. McCreight: Classes of computable functions defined by bounds of computations, *Proc. Symp. Th. Comp.* 1 (1969) 79-88.

[5] P. G. Odifreddi: *Classical Recursion Theory*, vol. II, North Holland, 1999.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il