A RECURSION-THEORETIC CHARACTERISATION OF THE POSITIVE POLYNOMIAL-TIME FUNCTIONS

ANUPAM DAS[1] AND ISABEL OITAVEM[2]

Introduction

Monotone functions abound in the theory of computation, e.g. sorting a string, and detecting cliques in graphs. They have been comprehensively studied in the setting of *circuit complexity*, via $\neg$-free circuits (usually called 'monotone circuits'), cf. [Kor03]. Indeed several seminal results in circuit complexity concern bounds on the size of $\neg$-free circuits, e.g. [Raz85, AB87, Tar88].

The study of 'uniform' monotone computation is a much less developed subject. Grigni and Sipser began a line of work studying the effect of restricting 'negation' in computational models [GS92, Gri91]. One shortfall of their work was that deterministic classes lacked a bona fide treatment, with positive models only natively defined for nondeterminstic classes. This means that positive versions of, say, **P** must rather be obtained via indirect characterisations, e.g. as **ALOGSPACE**.

Later work by Lautemann, Schwentick and Stewart solved this problem by proposing a model of deterministic computation whose polynomial-time predicates coincide with several characterisations of **P** once 'negative' operations are omitted [LSS96, LSS98]. This induces a robust definition of a class '**posP**', the *positive* polynomial-time predicates [GS92, Gri91].

Here we extend this line of work to associated function classes, which are of natural interest for logical approaches to computational complexity, e.g. [Bus86, CN10] (see, e.g., [CK02]). Noting that several of the characterisations proposed by [LSS96] make sense for function classes (and, indeed, coincide), we propose a *function algebra* for the 'positive polynomial-time functions' on binary words (**posFP**) based on Cobham's bounded recursion on notation [Cob65]. We show that this algebra indeed coincides with certain characterisations proposed in [LSS96].

This work is based on the submitted preprint [DO18].

Monotone functions and positive computation

We consider binary strings (or 'words'), i.e. elements of $\{0,1\}^* = \bigcup_{n\in\mathbb{N}} \{0,1\}^n$, and for $x \in \{0,1\}^n$ we write $x(j)$ for the j^{th} bit of x, where $j = 0, \ldots, n-1$. We follow the usual convention that bits are indexed from right ('least significant') to left ('most significant'), e.g. as in [CK02]; for instance the word 011 has 0^{th} bit

[1]University of Copenhagen

[2]CMA and DM, FCT, Universidade Nova de Lisboa

[1]This work has been supported by a Marie Skłodowska-Curie fellowship, *Monotonicity in Logic and Complexity*, ERC project 753431.

[2]This work is partially supported by the Portuguese Science Foundation, FCT, through the projects UID/MAT/00297/2013 and PTDC/MHC-FIL/2583/2014.

1, 1st bit 1 and 2nd bit 0. We write $\varepsilon, \mathsf{s}_0, \mathsf{s}_1$ for the usual generators of $\{0,1\}^*$, with ε denoting the empty string, $\mathsf{s}_0 x = x0$ and $\mathsf{s}_1 x = x1$. We also write 1^n for 1 concatenated with itself n times, for $n \in \mathbb{N}$.

We consider functions of type $\{0,1\}^* \times \cdots \times \{0,1\}^* \to \{0,1\}^*$. For $n \in \mathbb{N}$, we define $\leq^n$ as the n-wise product order of $\leq$ on $\{0,1\}$, i.e. for $x, y \in \{0,1\}^n$ we have $x \leq^n y$ if $\forall j < n.\ x(j) \leq y(j)$. The partial order $\leq$ on $\{0,1\}^*$ is the union of all $\leq^n$, for $n \in \mathbb{N}$. A function $f : (\{0,1\}^*)^k \to \{0,1\}^*$ is *monotone* if $x_1 \leq y_1, \ldots, x_k \leq y_k \implies f(\vec{x}) \leq f(\vec{y})$. One particular feature of monotone functions, independent of any machine model, is that they are rather oblivious: the length of the output depends only on the length of the inputs:

Observation 1. *Let $f(x_1, \ldots, x_k)$ be a monotone function. Then, whenever $|x_1| = |y_1|, \ldots, |x_k| = |y_k|$, also $|f(\vec{x})| = |f(\vec{y})|$.*

Uniform families of ¬-free circuits

One way to define a positive variant of **FP** is to consider ¬-free circuits that are in some sense uniform. [LSS96, LSS98] followed this approach too for **P**, showing that one of the strongest levels of uniformity (**P**) and one of the weakest levels ('quantifier-free') needed to characterise **P** indeed yield the same class of languages when describing ¬-free circuits. We consider Δ_0-uniformity rather than quantifier-free uniformity in [LSS96, LSS98] since it is easier to present and suffices for our purposes. (We point out that this subsumes, say, **L**-uniformity.)

Recall that a Δ_0 formula is a first-order formula over $\{0, 1, +, \times, <\}$ where all quantifiers of the form $\exists x < t$ or $\forall x < t$ for a term t. A Δ_0-formula $\varphi(n_1, \ldots, n_k)$ is interpreted over $\mathbb{N}$ in the usual way, computing the set $\{\vec{n} \in \mathbb{N}^k : \mathbb{N} \vDash \varphi(\vec{n})\}$.

Definition 2 (Positive circuits)**.** A family of *k-argument ¬-free circuits* is a set $\{C(\vec{n})\}_{\vec{n} \in \mathbb{N}^k}$, where each $C(\vec{n})$ is a circuit with arbitrary fan-in $\bigvee$ and $\bigwedge$ gates,[1] given as a tuple $(N, D, E, I_1, \ldots, I_k, O)$, where $[N] = \{n < N\}$ is the set of *gates*, $D \subseteq [N]$ is the set of $\bigvee$ gates (remaining gates are assumed to be $\bigwedge$), $E \subseteq [N] \times [N]$ is the set of (directed) edges (requiring $E(m,n) \implies m < n$), $I_j \subseteq [n_j] \times [N]$ contains just pairs (l, n) s.t. the l^{th} bit of the j^{th} input is connected to the gate n, and $O \subseteq [N]$ is the (ordered) set of output gates.

If these sets are polynomial-time computable from inputs $(1^{n_1}, \ldots, 1^{n_k})$ then we say the circuit family is **P**-*uniform*. Similarly, we say the family is Δ_0-*uniform* if $N(\vec{n})$ is a term (i.e. a polynomial) in $\vec{n}$ and there are Δ_0-formulae $D(n, \vec{n})$, $E(m, n, \vec{n})$, $I_j(l, n, \vec{n})$, $O(n, \vec{n})$ computing the associated sets.

Notice that, importantly, we restrict the set O of output gates to depend only on the length of the inputs, not their individual bit-values; this is pertinent thanks to Obs. 1. Also, when it is convenient, we may construe I_j as a function $[n_j] \to \mathcal{P}([N])$, by Currying.

Positive Turing machines

Now we introduce a machine model for positive computation. The definition of a multitape machine below is essentially from [Pap07]. The monotonicity criterion is identical to that from [LSS96, LSS98], though we also allow auxiliary 'work' tapes so that the model is easier to manipulate. This also means that we do not need

[1] Note that a $\bigvee$ gate with zero inputs outputs 0, while a $\bigwedge$ gate with zero inputs outputs 1.

explicit accepting and rejecting states with the further monotonicity requirements from [LSS96, LSS98], since this is subsumed by the monotonicity requirement on writing 0s and 1s: predicates can be computed in the usual way by Boolean valued functions, with 0 indicating 'reject' and 1 indicating 'accept'.

Definition 3 (Positive machines)**.** A k-tape (deterministic) *Turing machine* (TM) is a tuple $M = (Q, \Sigma, \delta, s, h)$ such that:

- Q is a finite set of (non-final) *states*.
- $\Sigma \supseteq \{\triangleright, \square, 0, 1\}$ is a finite set, called the *alphabet*.
- $\delta : Q \times \Sigma^k \to (Q \cup \{h\}) \times (\Sigma \times \{\leftarrow, -, \rightarrow\})^k$ such that, whenever $\delta(q, \sigma_1, \ldots, \sigma_k) = (q, \tau_1, d_1, \ldots, \tau_k, d_k)$, if $\sigma_i = \triangleright$ then $\tau_i = \triangleright$ and $d_i = \rightarrow$.
- $s \in Q$ is the *initial* state.
- Q and Σ are disjoint, and neither contains the symbols $h, \leftarrow, -, \rightarrow$.

We call h the *final* state, $\triangleright$ the 'beginning of tape marker', $\square$ the 'blank' symbol, and $\leftarrow, -, \rightarrow$ are the *directions* 'left', 'stay' and 'right'.

Now, write $\mathcal{I} = Q \times \Sigma^k$ and $\mathcal{O} = (Q \cup \{h\}) \times (\Sigma \times \{\leftarrow, -, \rightarrow\})^k$, so that δ is a function $\mathcal{I} \to \mathcal{O}$. We define partial orders $\leq_{\mathcal{I}}$ and $\leq_{\mathcal{O}}$ on $\mathcal{I}$ and $\mathcal{O}$ resp. as follows:

- $(q, \sigma_1, \ldots, \sigma_k) \leq_{\mathcal{I}} (q', \sigma'_1, \ldots, \sigma'_k)$ if $q = q'$ and, for $i = 1, \ldots, k$, either $\sigma_i = \sigma'_i$, or both $\sigma_i = 0$ and $\sigma'_i = 1$.
- $(q, \sigma_1, d_1, \ldots, \sigma_k, d_k) \leq_{\mathcal{O}} (q', \sigma'_1, d'_1, \ldots, \sigma'_k, d'_k)$ if $q = q'$ and, for $i = 1, \ldots, k$, we have $d_i = d'_i$ and either $\sigma_i = \sigma'_i$, or both $\sigma_i = 0$ and $\sigma'_i = 1$.

We say that M is *positive* (a PTM) if $\delta : \mathcal{I} \to \mathcal{O}$ is monotone with respect to $\leq_{\mathcal{I}}$ and $\leq_{\mathcal{O}}$, i.e. $I \leq_{\mathcal{I}} I' \implies \delta(I) \leq_{\mathcal{O}} \delta(I')$.

A *run* of input strings $x_1, \ldots, x_k \in \{0,1\}^*$ on M is defined in the usual way (see, e.g., [Pap07]), beginning from the initial state s and initialising the i^{th} tape to $\triangleright x_i \square^\infty$, for $i = 1, \ldots, k$. If M halts, i.e. reaches the state h, its *output* is whatever is printed on the k^{th} tape at that moment, up to the first $\square$ symbol.

We say that a function $f : (\{0,1\}^*)^k \to \{0,1\}^*$ is *computable by a PTM* if there is a k'-tape PTM M, with $k' \geq k$, such that M halts on every input and, for inputs $(x_1, \ldots, x_k, \varepsilon, \ldots, \varepsilon)$, outputs $f(x_1, \ldots, x_k)$.

The monotonicity condition on the transition function above means that the value of a Boolean read does not affect the next state or cursor movements (this reflects the 'obliviousness' of monotone functions, cf. Prop. 1). Moreover, it may only affect the *Boolean* symbols printed: the machine may read 0 and print 0 but read 1 and print 1, in otherwise-the-same situation. However, if in one situation it prints a non-Boolean σ when reading a Boolean i, it also prints σ when reading $\neg i$.

A uniform version of Cobham's algebra for **FP**

We present a *function algebra* for positive feasible computation by considering 'uniform' versions of recursion operators. We write $[\mathcal{F}; \mathcal{O}]$ for the function class generated by a set of initial functions $\mathcal{F}$ and a set of operations $\mathcal{O}$, and generally follow conventions and notations from [CK02].

Let us first recall Cobham's function algebra for the polynomial-time functions, **FP**. This algebra was originally formulated over natural numbers, though the version over binary words here is essentially as in [Fer90] and [Oit97]. Define $\pi^k_j(x_1, \ldots, x_k) := x_j$ and $x \# y := 1^{|x||y|}$. We also write comp for the operation of function composition.

Definition 4. A function $f(x, \vec{x})$ is defined by *bounded recursion on notation* (BRN) from g, h_0, h_1, k if $|f(x, \vec{x})| \leq |k(x, \vec{x})|$ for all $x, \vec{x}$ and:

$$\begin{array}{rcl} f(\varepsilon, \vec{x}) & = & g(\vec{x}) \\ f(\mathsf{s}_0 x, \vec{x}) & = & h_0(x, \vec{x}, f(x, \vec{x})) \\ f(\mathsf{s}_1 x, \vec{x}) & = & h_1(x, \vec{x}, f(x, \vec{x})) \end{array} \tag{1}$$

We write C for the function algebra $[\varepsilon, \mathsf{s}_0, \mathsf{s}_1, \pi^k_j, \#; \mathsf{comp}, \mathsf{BRN}]$.

Theorem 5 ([Cob65])**.** $\mathsf{C} = \mathbf{FP}$.

Notice that $\varepsilon, \mathsf{s}_0, \mathsf{s}_1, \pi^k_j, \#$ are monotone, and the composition of two monotone functions is again monotone. However, non-monotone functions are definable using BRN, for instance:

$$\begin{array}{rcl} cond(\varepsilon, y_\varepsilon, y_0, y_1) & = & y_\varepsilon \\ cond(\mathsf{s}_0 x, y_\varepsilon, y_0, y_1) & = & y_0 \\ cond(\mathsf{s}_1 x, y_\varepsilon, y_0, y_1) & = & y_1 \end{array} \tag{2}$$

This 'conditional' function is definable since we do not force any connection between h_0 and h_1 in (1). Insisting on $h_0 \leq h_1$ would retain monotonicity, but this condition is external and not generally checkable. Instead, we can impose monotonicity implicitly by somewhat 'uniformising' BRN. First, we will need to recover certain monotone variants of the conditional:

Definition 6 (Meets and joins)**.** We define $x \wedge y = z$ by $|z| = \min(|x|, |y|)$ and $z(j) = \min(x(j), y(j))$, for $j < \min(|x|, |y|)$. We define analogously $x \vee y = z$ by $|z| = \max(|x|, |y|)$ and $z(j) = \max(x(j), y(j))$, for $j < \min(|x|, |y|)$

Definition 7 (The function algebra $u\mathsf{C}$)**.** We say that a function is defined by *uniform bounded recursion on notation* ($u\mathsf{BRN}$) from g, h, k if $|f(x, \vec{x})| \leq |k(x, \vec{x})|$ for all $x, \vec{x}$ and:

$$\begin{array}{rcl} f(\varepsilon, \vec{x}) & = & g(\vec{x}) \\ f(\mathsf{s}_0 x, \vec{x}) & = & h(0, x, \vec{x}, f(x, \vec{x})) \\ f(\mathsf{s}_1 x, \vec{x}) & = & h(1, x, \vec{x}, f(x, \vec{x})) \end{array} \tag{3}$$

We define $u\mathsf{C}$ to be the function algebra $[\varepsilon, \mathsf{s}_0, \mathsf{s}_1, \pi^k_j, \#, \wedge, \vee; \mathsf{comp}, u\mathsf{BRN}]$.

Notice that $\wedge$ and $\vee$ are clearly $\mathbf{FP}$ functions, therefore they are in C. Moreover, notice that ($u\mathsf{BRN}$) is the special case of (1) when $h_i(x, \vec{x}, y)$ has the form $h(i, x, \vec{x}, y)$. So, we have that $u\mathsf{C} \subseteq \mathsf{C} = \mathbf{FP}$.

Main results

Our main result is that all the previously introduced models of computation coincide, in terms of the polynomial-time 0-1 functions they compute:

Theorem 8. *The following function classes are equivalent:*

(1) Functions on $\{0,1\}^*$ computable by Δ_0-uniform families of $\neg$-free circuits.
(2) Functions on $\{0,1\}^*$ computable by multitape PTMs in polynomial time.
(3) Functions on $\{0,1\}^*$ definable in $u\mathsf{C}$.
(4) Functions on $\{0,1\}^*$ computable by $\mathbf{P}$-uniform families of $\neg$-free circuits.

This result is similar to analogous ones found in [LSS96] for positive versions of the predicate class **P**. The proof, which can be found in [DO18], uses mostly standard techniques, adapted to the monotone setting. Notice that the equivalence of models thus holds for any level of uniformity between Δ_0 and **P**, e.g. for **L**-uniform ¬-free circuits.

Finally, the robustness of the class induced by Thm. 8 above allows us to recover a natural notion of 'positive polynomial-time function':

Definition 9 (Positive **FP**)**.** We define the function class **posFP** as the set of functions on $\{0,1\}^*$ computed by any of the equivalent models from Thm. 8.

In [DO18] we furthermore give a function algebra based on *safe recursion*, in the style of Bellantoni and Cook [BC92], yielding an entirely *implicit* characterisation of **posFP**, mentioning neither explicit bounds nor explicit monotonicity constraints. As far as we know, this is the first implicit approach to monotone computation.

References

[AB87] Noga Alon and Ravi B Boppana. The monotone circuit complexity of boolean functions. *Combinatorica*, 7(1):1–22, 1987.

[BC92] Stephen Bellantoni and Stephen A. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.

[Bus86] Samuel R. Buss. *Bounded arithmetic*, volume 1 of *Studies in Proof Theory*. Bibliopolis, Naples, 1986.

[CK02] Peter Clote and Evangelos Kranakis. *Boolean Functions and Computation Models*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2002.

[CN10] Stephen Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[Cob65] A. Cobham. The intrinsic computational difficulty of functions. In *Proc. of the International Congress for Logic, Methodology, and the Philosophy of Science*, pages 24–30. Amsterdam, 1965.

[DO18] A. Das and I. Oitavem. A recursion-theoretic characterisation of the positive polynomial-time functions, 2018. Submitted. `http://www.anupamdas.com/wp/pos-fp/`.

[Fer90] Fernando Ferreira. Polynomial time computable arithmetic. In *Contemporary Mathematics*, volume 106, pages 137–156. AMS, 1990.

[Gri91] Michelangelo Grigni. *Structure in monotone complexity*. PhD thesis, 1991.

[GS92] Michelangelo Grigni and Michael Sipser. Monotone complexity. In *London Mathematical Society Symposium on Boolean Function Complexity*, New York, NY, USA, 1992. Cambridge University Press.

[Kor03] A D Korshunov. Monotone boolean functions. *Russian Mathematical Surveys*, 58(5), 2003.

[LSS96] Clemens Lautemann, Thomas Schwentick, and Iain A. Stewart. On positive P. In *IEEE Conference on Computational Complexity '96*, 1996.

[LSS98] Clemens Lautemann, Thomas Schwentick, and Iain A. Stewart. Positive versions of polynomial time. *Inf. Comput.*, 147(2):145–170, 1998.

[Oit97] Isabel Oitavem. New recursive characterizations of the elementary functions and the functions computable in polynomial space. *Revista Matematica de la Universidad Complutense de Madrid*, 10(1):109–125, 1997.

[Pap07] Christos H. Papadimitriou. *Computational complexity*. Academic Internet Publ., 2007.

[Raz85] A. A. Razborov. Lower bounds on the monotone complexity of some Boolean functions. *Doklady Akademii Nauk SSSR*, 285, 1985.

[Tar88] E. Tardos. The gap between monotone and non-monotone circuit complexity is exponential. *Combinatorica*, 8(1):141–142, 1988.