A recursion-theoretic characterisation of the positive polynomial-time functions

Das, Anupam; Oitavem, Isabel

DOI:
10.4230/LIPIcs.CSL.2018.18

License:
Creative Commons: Attribution (CC BY)

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (Harvard):
Das, A & Oitavem, I 2018, A recursion-theoretic characterisation of the positive polynomial-time functions. in DR Ghica & A Jung (eds), *27th EASCL Annual Conference on Computer Science Logic 2018 (CSL 2018).*, 18, Leibniz International Proceedings in Informatics, LIPIcs, vol. 119, Schloss Dagstuhl, 27th Annual EACSL Conference Computer Science Logic, CSL 2018, Birmingham, United Kingdom, 4/09/18. https://doi.org/10.4230/LIPIcs.CSL.2018.18

A Recursion-Theoretic Characterisation of the Positive Polynomial-Time Functions

Anupam Das[1]
University of Copenhagen, Denmark
anupam.das@di.ku.dk

Isabel Oitavem[2]
CMA and DM, FCT, Universidade Nova de Lisboa, Portugal
oitavem@fct.unl.pt

Abstract

We extend work of Lautemann, Schwentick and Stewart [14] on characterisations of the "positive" polynomial-time predicates (**posP**, also called **mP** by Grigni and Sipser [11]) to function classes. Our main result is the obtention of a function algebra for the positive polynomial-time functions (**posFP**) by imposing a simple uniformity constraint on the bounded recursion operator in Cobham's characterisation of **FP**. We show that a similar constraint on a function algebra based on *safe recursion*, in the style of Bellantoni and Cook [3], yields an "implicit" characterisation of **posFP**, mentioning neither explicit bounds nor explicit monotonicity constraints.

2012 ACM Subject Classification Theory of computation → Recursive functions

Keywords and phrases Monotone complexity, Positive complexity, Function classes, Function algebras, Recursion-theoretic characterisations, Implicit complexity, Logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.18

Acknowledgements The authors would like to thank Patrick Baillot, Sam Buss, Reinhard Kahle and the anonymous reviewers for several helpful discussions on this subject.

1 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 ¬-free circuits (usually called "monotone circuits"), cf. [13]. Most notably, Razborov's seminal work [20] gave exponential lower bounds on the size of monotone circuits, and later refinements, cf. [1, 23], separated them from non-monotone circuits altogether.

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 [11, 10]. 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

[1] This work was 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.

27th EACSL Annual Conference on Computer Science Logic (CSL 2018).
Editors: Dan Ghica and Achim Jung; Article No. 18; pp. 18:1–18:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

with several characterisations of **P** once "negative" operations are omitted [14, 15]. This induces a robust definition of a class "**posP**", the *positive* polynomial-time predicates [11, 10].

In this paper we extend this line of work to associated function classes (see, e.g., [5]), which are of natural interest for logical approaches to computational complexity, e.g. [4, 7]. Noting that several of the characterisations proposed by [14] 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* [6]. We show that this algebra indeed coincides with certain characterisations proposed in [14], and furthermore give a function algebra based on *safe recursion*, in the style of Bellantoni and Cook [3]. The latter constitutes 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.

This paper is structured as follows. In Sect. 2 we present preliminaries on monotone functions on binary strings and recall some notions of positive computation from [14, 15]. We show also that these models compute the same class of functions (Thm. 7), inducing our definition of **posFP**. In Sect. 3 we recall Cobham's function algebra for **FP**, based on bounded recursion on notation, and introduce a uniform version of it, $u\mathsf{C}$, which we show is contained in **posFP** in Sect. 4 (Thm. 17). In Sect. 5 we prove some basic properties about $u\mathsf{C}$; we characterise the *tally* functions of $u\mathsf{C}$, those that return unary outputs on unary inputs, as just the unary codings of linear space functions on $\mathbb{N}$, by giving an associated function algebra (Thm. 21). We use this to recover a proof that $u\mathsf{C}$ is closed under a *simultaneous* version of its recursion scheme (Thm. 28), tracking the length of functions rather than usual methods relying on explicit pairing functions. In Sect. 6 we show the converse result that $u\mathsf{C}$ contains **posFP** (Thm. 30). Finally, in Sect. 7 we give a characterisation of **posFP** based on "safe" recursion (Thm. 36), and we give some concluding remarks in Sect. 8.

Throughout this work, we follow the convention of [14, 15], reserving the word "monotone" for the semantic level, and rather using "positive" to describe restricted models of computation.

2 Monotone functions and positive computation

We consider binary strings (or "words"), i.e. elements of $\{0,1\}^* = \bigcup\limits_{n\in\mathbb{N}} \{0,1\}^n$, 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 [5]; for instance the word 011 has 0^{th} bit 1, 1^{st} bit 1 and 2^{nd} bit 0.

We write $\varepsilon, \mathsf{s}_0, \mathsf{s}_1$ for the usual generators of $\{0,1\}^*$, i.e. ε denotes 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 $x_1 \leq y_1, \ldots, x_k \leq y_k \implies f(\vec{x}) \leq f(\vec{y})$.

▶ **Example 1.** A recurring example we will consider is the *sorting* function $sort(x)$, which takes a binary word input and rearranges the bits so that all 0s occur before all 1s, left-right. Clearly *sort* is monotone, and can be given the following recursive definition:

$$\begin{array}{rcl} sort(\varepsilon) & = & \varepsilon \\ sort(\mathsf{s}_0 x) & = & 0 sort(x) \\ sort(\mathsf{s}_1 x) & = & sort(x) 1 \end{array} \tag{1}$$

While in the binary case it may seem rather simple, we will see that *sort* nonetheless exemplifies well the difference between positive and non-positive computation.

One particular well-known 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 2.** *Let $f(x_1, \dots, x_k)$ be a monotone function. Then, whenever $|x_1| = |y_1|, \dots, |x_k| = |y_k|$, we also have that $|f(\vec{x})| = |f(\vec{y})|$.*

Proof. Let $n_j = |x_j| = |y_j|$, for $1 \leq j \leq k$. We have both $f(\vec{x}) \leq f(1^{n_1}, \dots, 1^{n_k})$ and $f(\vec{y}) \leq f(1^{n_1}, \dots, 1^{n_k})$, by monotonicity, so indeed all these outputs have the same length. ◀

One way to define a positive variant of **FP** is to consider ¬-free circuits that are in some sense uniform. [14, 15] 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 show that a similar result holds for classes of functions, when allowing circuits to have many output wires. Most of the techniques used in this section are standard, so we keep to a high-level exposition, rather dedicating space to examples of the notions of positive computation presented.

We consider Δ_0-uniformity rather than quantifier-free uniformity in [14, 15] since it is easier to present and suffices for our purposes. (We point out that this subsumes, say, **L**-uniformity, as explained in the Remark below.) 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 $\varphi(n_1, \dots, n_k)$ is interpreted over $\mathbb{N}$ in the usual way, and naturally computes the set $\{\vec{n} \in \mathbb{N}^k : \mathbb{N} \vDash \varphi(\vec{n})\}$.

▶ **Definition 3** (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,[3] given as a tuple $(N, D, E, I_1, \dots, I_k, O)$, 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 $(1^{n_1}, \dots, 1^{n_k})$ 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.

The specification of a circuit family above is just a variant of the usual "direct connection language" from circuit complexity, cf. [22]. 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 Prop. 2. Also, when it is convenient, we may construe I_j as a function $[n_j] \to \mathcal{P}([N])$, by Currying.

▶ Remark. Δ_0-sets are well known to be complete for the linear-time hierarchy [24]. However, since we only need to manipulate "unary" inputs in the notion of Δ_0-uniformity above, the circuits generated are actually **LH**-uniform, where **LH** is the logarithmic-time hierarchy, the uniform version of $\mathbf{AC}^0$ [2]. See, e.g., [5] Sect. 6.3 for related discussions on **LH** and, e.g., [7] Sect. IV.3 for some relationships between Δ_0 and $\mathbf{AC}^0$.

[3] By convention, a $\bigvee$ gate with zero inputs outputs 0, while a $\bigwedge$ gate with zero inputs outputs 1.

▶ **Example 4** (Circuits for sorting). Let us write $th(j,x)$ for the $(j-1)^{\mathrm{th}}$ bit of $sort(x)$, for $1 \leq j \leq |x|$. We also set $th(0,x) = 1$ and $th(j,x) = 0$ for $j > |x|$. Notice that $th(j,x) = 1$ precisely if there are at least j 1s in x, i.e. it is a *threshold* function. We assume that the input j is given in unary, for monotonicity, but as an abuse of notation write, say, j rather than 1^j throughout this example to lighten the notation. (Later, in Sect. 5, we will be more formal when handling unary inputs.)
We have the following recurrence, for $j > 0$:

$$th(j, \mathsf{s}_i x) = th(j,x) \vee (i \wedge th(j-1,x)) \tag{2}$$

Notice that this recurrence treats the $i = 0$ and $i = 1$ cases in the "same way". This corresponds to the notion of *uniformity* that we introduce in our function algebras later. We can use this recurrence to construct polynomial-size $\neg$-free circuits for sorting. For an input x of size n, write x^l for the prefix $x(l-1) \cdot \cdots \cdot x(0)$. Informally, we construct a circuit with $n+1$ "layers" (numbered $0, \ldots, n$), where the l^{th} layer outputs $th(n, x^l) \cdot \cdots \cdot th(0, x^l)$; the layers are connected to each other according to the recurrence in (2), with $th(0, x^l)$ always set to 1. Each layer will thus have $2(n+1)$ gates, with $(n+1)$ disjunction gates (computing the functions $th(j, x^l)$), and $n+1$ intermediate conjunction gates. We assign odd numbers to disjunction gates and even numbers to conjunction gates, so that the total number of gates is $N(n) = 2(n+1)^2$ and $D(n) = \{2r+1 : r < (n+1)^2\}$. The sets $E(r,s,n)$ and $I(r,n)$ can be given a routine description, and the set $O(r,n)$ of output gates consists of just the final layer of disjunction gates (except the rightmost), computing $th(n,x) \cdot \cdots \cdot th(1,x)$, i.e. $O(r,n) = \{2(n+1)^2 - 2r - 1 : r < n\}$. It is not hard to see that such circuits are not only $\mathbf{P}$-uniform, but also Δ_0-uniform.

Now we introduce a machine model for uniform positive computation. The definition of a multitape machine below is essentially from [19]. The monotonicity criterion is identical to that from [14, 15], though we also allow auxiliary "work" tapes so that the model is easier to manipulate. This also means that we do not need explicit accepting and rejecting states with the further monotonicity requirements from [14, 15], 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 5** (Positive machines). A k-tape (deterministic) *Turing machine* (TM) is a tuple $M = (Q, \Sigma, \delta, s, h)$ where:

- 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 $x_1, \dots, x_k \in \{0,1\}^*$ on M is defined in the usual way (see, e.g., [19]), beginning from the initial state s and initialising the i^{th} tape to $\rhd x_i \Box^\omega$, for $i = 1, \dots, k$. 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 $\Box$ 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 $(x_1, \dots, x_k, \varepsilon, \dots, \varepsilon)$, outputs $f(x_1, \dots, x_k)$.

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. 2). 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 0 or 1, it must also print σ when reading the other.

▶ **Example 6** (Machines for sorting). A simple algorithm for sorting a binary string x is as follows: do two passes of x, first copying the 0s in x onto a fresh tape, then appending the 1s.[4] However, it is not hard to see that a machine directly implementing this algorithm will not be positive. Instead, we may again use the recurrence from (2).

We give an informal description of a PTM that sorts a binary string. The machine has four tapes; the first is read-only and stores the input, say x with $|x| = n$. As in Ex. 4, we inductively compute $t^l = \mathit{th}(n, x^l) \cdot \dots \cdot \mathit{th}(0, x^l)$, for $l \leq n$. The second and third tape are used to temporarily store t^l, while the fourth is used to compute the sorting of the next prefix t^{l+1}. At each step the cursors on the working tapes move to the next bit and the transition function implements the recurrence from (2), calculating the next bit of t^{l+1} and writing it to the fourth tape. Notice that the cursor on the third tape remains one position offset from the cursor on the second and fourth tapes, cf. (2). Once t^{l+1} has been completely written on the fourth tape the machine copies it over the contents of the second and third tapes and erases the fourth tape before moving onto the next bit of the first tape and repeating the process. Finally, once the first tape has been exhausted, the machine copies the contents of the second (or third) tape, except the last bit (corresponding to $\mathit{th}(x, 0) = 1$), onto the fourth tape and halts.

▶ **Theorem 7.** *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 multi-tape PTMs that halt in polynomial time.*
(3) *Functions on $\{0,1\}^*$ computable by $\mathbf{P}$-uniform families of $\neg$-free circuits.*
This result is similar to analogous ones found in [14] for positive versions of the predicate class $\mathbf{P}$. It uses standard techniques so we give only a sketch of the proof below. Notice that the equivalence of models thus holds for any level of uniformity between Δ_0 and $\mathbf{P}$, e.g. for $\mathbf{L}$-uniform $\neg$-free circuits, cf. the Remark on p. 3.

Proof sketch of Thm. 7. We show that $(1) \subseteq (2) \subseteq (3) \subseteq (1)$. The containments are mostly routine, though $(3) \subseteq (1)$ requires some subtlety due to the positivity condition on circuits. For this we rely on an observation from [10]. Let $C(\vec{n})$ be a $\mathbf{P}$-uniform family of $\neg$-free

[4] Recall that, while bits are indexed from right to left, machines read from left to right.

circuits, specified by polynomial-time programs $N, D, E, I_1, \dots, I_k, O$. Since the circuit-value problem is **P**-complete under even $\mathbf{AC}^0$-reductions (see, e.g., [7]), we may recover Δ_0-uniform polynomial-size circuits (with negation) computing each of $N, D, E, I_1, \dots, I_k, O$, cf. the Remark on p. 3. However, these circuits take only unary strings of 1s as inputs, and so all negations can be pushed to the bottom (by De Morgan laws) and eliminated, yielding input-free $\neg$-free circuits for each of $N, D, E, I_1, \dots, I_k, O$ and their complements (by dualising gates). We may use these as "subcircuits" to compute the relevant local properties of $C(\vec{n})$. In particular, every internal gate n of $C(\vec{n})$ may be replaced by the following configuration (progressively, beginning from the highest-numbered gate $N(\vec{n}) - 1$):

$$\begin{aligned} &\left(D(n,\vec{n}) \ \wedge \ \left(\bigvee_{m<n} (m \wedge E(m,n,\vec{n})) \ \vee \ \bigvee_{j=1}^{k} \bigvee_{l<n_j} (x(l) \ \wedge \ I_j(l,n)) \right) \right) \\ \vee \ &\left(\neg D(n,\vec{n}) \ \wedge \ \left(\bigwedge_{m<n} (m \vee \neg E(m,n,\vec{n})) \ \wedge \ \bigwedge_{j=1}^{k} \bigwedge_{l<n_j} (x(l) \vee \neg I_j(l,n)) \right) \right) \end{aligned}$$

This entire construction can be made Δ_0-uniform, upon a suitable renumbering of gates.

The proof of (2) $\subseteq$ (3) follows a standard construction (see, e.g., [19]), observing that the positivity criterion on PTMs entails local monotonicity and hence allows us to construct circuits that are $\neg$-free. (Similar observations are made in [11, 10, 14, 15]). Suppose Q, Σ and $\{\leftarrow, -, \rightarrow\}$ are encoded by Boolean strings such that distinct elements are incomparable under $\leq$, (except $0 \leq 1$ for $0, 1 \in \Sigma$). Thus we may construe δ as a bona fide monotone Boolean function of fixed input arities, and thus has some (constant-size) $\neg$-free circuit thanks to adequacy of the basis $\{\bigvee, \bigwedge\}$, say C_δ. Now, on a fixed input, consider "configurations" of the form $(q, x_1, n_1, \dots, x_k, n_k)$, where $q \in Q$, x_i is the content of the i^{th} tape (up to the halting time bound) and n_i is the associated cursor position (encoded in unary). We may use C_δ to construct polynomial-size $\neg$-free circuits mapping the machine configuration at time t to the configuration at time $t + 1$. By chaining these circuits together polynomially many times (determined by the halting time bound), we may thus obtain a circuit that returns the output of the PTM. This entire construction remains **P**-uniform, as usual.

The proof of (1) $\subseteq$ (2) is also routine, building a PTM "evaluator" for $\neg$-free circuits, where $\neg$-freeness allows us to satisfy the positivity condition on TMs. We rely on the fact that the Δ_0-specifications may be entirely encoded in *unary* on a PTM, so that they are monotone, in polynomial-time. We do not go into details here since, in particular, this containment is subsumed by our later results, Thm. 17 and Thm. 30, which show that (1) $\subseteq$ uC $\subseteq$ (2), for the algebra uC we introduce in the next section. ◀

▶ **Definition 8** (Positive **FP**). The function class **posFP** is defined to be the set of functions on $\{0,1\}^*$ computed by any of the equivalent models from Thm. 7.

▶ Remark. The notion of *positive* computation was previously studied in [11, 14, 15]. One interesting point already noted in those works is that, for a complexity class, its positive version is not, in general, just its monotone members. This follows from a seminal result of Razborov [20], and later improvements [1, 23]: there are polynomial-time monotone predicates (and hence polynomial-size circuits with negation) for which the only $\neg$-free circuits are exponential in size. In particular, **posFP** $\subsetneq \{f \in \mathbf{FP} : f \text{ monotone}\}$.

3 An algebra uC for posFP

We present a *function algebra* for **posFP** 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 [5].

Let us first recall Cobham's function algebra for the polynomial-time functions, **FP**. This algebra was originally formulated over natural numbers, though we work with a version here over binary words, essentially as in [9, 18].

Define $\pi^k_j(x_1, \ldots, x_k) := x_j$ and $x\#y := 1^{|x||y|}$. We write comp for the operation of function composition.

▶ **Definition 9.** A function f 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:

$$\begin{array}{rcl} f(\varepsilon,\vec{x}) & = & g(\vec{x}) \\ f(\mathsf{s}_0 x,\vec{x}) & = & h_0(x,\vec{x},f(x,\vec{x})) \\ f(\mathsf{s}_1 x,\vec{x}) & = & h_1(x,\vec{x},f(x,\vec{x})) \end{array} \tag{3}$$

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

▶ **Theorem 10** ([6])**.** $\mathsf{C} = \mathbf{FP}$.

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:

$$\begin{array}{rcl} \mathit{cond}(\varepsilon, y_\varepsilon, y_0, y_1) & = & y_\varepsilon \\ \mathit{cond}(\mathsf{s}_0 x, y_\varepsilon, y_0, y_1) & = & y_0 \\ \mathit{cond}(\mathsf{s}_1 x, y_\varepsilon, y_0, y_1) & = & y_1 \end{array} \tag{4}$$

This "conditional" function is definable since we do not force any connection between h_0 and h_1 in (3). 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 11** (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 $j < \max(|x|, |y|)$.

Note that, in the case of $x \vee y$ above, if $|x| < |y|$ and $|x| \leq j < \max(|x|, |y|)$, then $x(j)$ is not defined and we set $z(j) = y(j)$. We follow an analogous convention when $|y| < |x|$.

▶ **Definition 12** (The function algebra uC)**.** We say that a function is defined by *uniform bounded recursion on notation* (uBRN) from g, h, k if $|f(x,\vec{x})| \leq |k(x,\vec{x})|$ for all $x, \vec{x}$ and:

$$\begin{array}{rcl} f(\varepsilon,\vec{x}) & = & g(\vec{x}) \\ f(\mathsf{s}_0 x,\vec{x}) & = & h(0,x,\vec{x},f(x,\vec{x})) \\ f(\mathsf{s}_1 x,\vec{x}) & = & h(1,x,\vec{x},f(x,\vec{x})) \end{array} \tag{5}$$

We define uC 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 **FP** functions, therefore they are in C. Moreover, notice that (5) is the special case of (3) when $h_i(x, \vec{x}, y)$ has the form $h(i, x, \vec{x}, y)$. So, we have that $u\mathsf{C} \subseteq \mathsf{C} = \mathbf{FP}$. We will implicitly use this observation later to ensure that the outputs of uC functions have lengths which are polynomially bounded on the lengths of the inputs.

The main result of this work is that $u\mathsf{C} = \mathbf{posFP}$. The two directions of the equality are proved in the sections that follow, in the form of Thms. 17 and 30. Before that, we make some initial observations about uC.

▶ **Proposition 13.** $u\mathsf{C}$ *contains only monotone functions.*

Proof. The proof is by induction on the definition of f. The relevant case is when f is defined by $u\mathsf{BRN}$. It suffices to show that f is monotone in its first input, which we do by induction on its length. Let $w \leq x$. If $|w| = |x| = 0$, then they are both ε and we are done. Otherwise let $w = \mathsf{s}_i w'$ and $x = \mathsf{s}_j x'$. Then $f(w, \vec{y}) = h(i, w', \vec{y}, f(w', \vec{y})) \leq h(j, x', \vec{y}, f(x', \vec{y}))$ by the inductive hypothesis, since $i \leq j$ and $w' \leq x'$, and we are done. ◀

▶ **Proposition 14.** $u\mathsf{C} + \mathit{cond} = \mathsf{C}$.[5]

Proof. The left-right inclusion follows from the definition of *cond* by BRN in (4). For the right-left inclusion, we again proceed by induction on the definition of functions in C, and the relevant case is when f is defined by BRN, say from g, h_0, h_1, k. In this case, we may recover a definition of f using $u\mathsf{BRN}$ by writing $h(i, x, \vec{x}, y) = \mathit{cond}(i, g(\vec{x}), h_0(x, \vec{x}, y), h_1(x, \vec{x}, y))$. ◀

As expected, $u\mathsf{C}$ contains the usual predecessor function, least significant parts, concatenation, and a form of iterated predecessor:

▶ **Proposition 15** (Basic functions in $u\mathsf{C}$)**.** $u\mathsf{C}$ *contains the following functions:*[6]

$$\begin{array}{cccc} p(\varepsilon) := \varepsilon & \mathit{lsp}(\varepsilon) := \varepsilon & x \cdot \varepsilon := x & \mathit{msp}(|\varepsilon|, y) := y \\ p(\mathsf{s}_i x) := x & \mathit{lsp}(\mathsf{s}_i x) := i & x \cdot (\mathsf{s}_i y) := \mathsf{s}_i(x \cdot y) & \mathit{msp}(|\mathsf{s}_i x|, y) := p(\mathit{msp}(|x|, y)) \end{array}$$

Proof. All these definitions are instances of $u\mathsf{BRN}$, with bounding function $\#(\mathsf{s}_1 x, \mathsf{s}_1 y)$. ◀

Notice that, in the above definition of concatenation and throughout this work, we write $\mathsf{s}_i x$ for $\mathsf{s}_0 x \vee i$. We also sometimes simply write xy instead of $x \cdot y$.

We may also extract individual bits and test for the empty string in:

▶ **Proposition 16** (Bits and tests)**.** $u\mathsf{C}$ *contains the following functions:*

$$\mathit{bit}(|x|, y) := \mathit{lsp}(\mathit{msp}(|x|, y)) \qquad \begin{array}{r} \mathit{cond}_\varepsilon(\varepsilon, y, z) := y \\ \mathit{cond}_\varepsilon(\mathsf{s}_i x, y, z) := z \end{array}$$

4 posFP contains $u\mathsf{C}$

One direction of our main result follows by standard techniques:

▶ **Theorem 17.** $u\mathsf{C} \subseteq \mathbf{posFP}$.

It is not hard to see that one can extract (uniform) $\neg$-free circuits from a $u\mathsf{C}$ program, but we instead give a PTM for each function of $u\mathsf{C}$.

Proof sketch of Thm. 17. The proof is by induction on the function definitions. We prove that for all $f \in u\mathsf{C}$ there exists a PTM M_f computing f in polynomial time. For the initial functions the result is straightforward, and composition is routine.

We give the important case of when f is defined by $u\mathsf{BRN}$ from functions $g, h, k \in u\mathsf{C}$, as in (5); we will assume there are no side variables $\vec{x}$, for simplicity, though the general case is similar. Let $|f(x)| \leq b(|x|)$ for some polynomial $b(n)$ (since, in particular, $f(x) \in \mathsf{C} = \mathbf{FP}$). By the inductive hypothesis, there are PTMs M_g (with t tapes) and M_h (with $3 + u$ tapes)

[5] Here we write $[\mathcal{F}; \mathcal{O}] + f$ for the function algebra $[\mathcal{F}, f; \mathcal{O}]$.
[6] Notice that we could have equivalently defined $\mathit{lsp}(x)$ as $x \wedge 1$.

computing, respectively, g and h in time bounded by $p_g(n)$ and $p_h(1,m,n)$ for inputs of lengths n and $(1,m,n)$, respectively, for appropriate polynomials p_g, p_h. We assume that M_g and M_h halt scanning the first cell of each tape. In case of M_h we also assume that the content of tapes 1 and 2 are not changed during the computation (i.e. are read-only), and that the machine halts with the output in tape 3 with the other u tapes empty. We may define an auxiliary machine, M, with 3 tapes. Whenever the recursion input x is on tape 1, every time we run M, it writes the two first inputs of a call to h on tapes 2 and 3 and shifts the cursor in x one bit along. This means that a bit of x will be on tape 2 and a prefix of x, up to that bit, will be on tape 3.
Such M may be constructed so that it is a positive TM which works in time bounded by $2|x|+1$.

Now, we describe a positive TM M_f (with $3+u+t$ tapes) computing f as follows:

1. Run M_g (over the last t tapes of M_f);
2. Enter state s, run M (over tapes 1-3), and if M reaches state H, halt;
3. Run M_h (over tapes 2,3, $3+u+t$, and tapes 4 to $u+3$ of M_f, in this order);
4. Go to (2).

Each run of M shifts the cursor of the input tape one cell to the right, so, as expected, it halts after $|x|$ repetitions of the loop above, and hence operates in polynomial time. ◀

5 Some properties of the algebra uC

We conduct some "bootstrapping" in the algebra uC, both for self-contained interest and also for use later on to prove the converse of Thm. 17 in Sect. 6.

5.1 An algebra for lengths: tally functions of uC and linear space

We characterise the *tally functions* of uC, i.e. those with unary inputs and outputs, as just the unary codings of functions on $\mathbb{N}$ computable in linear space. We carry this argument out in a recursion-theoretic setting so that the exposition is more self-contained.

To distinguish functions on $\mathbb{N}$ from functions on $\{0,1\}^*$, we use variables m, n etc. to vary over $\mathbb{N}$. We will also henceforth write $\underline{n}$ for 1^n, to lighten the presentation when switching between natural numbers and binary words.

Further to Prop. 2, for functions in uC we may actually compute output lengths in a simple function algebra over $\mathbb{N}$.

▶ **Definition 18.** Let $0, 1, +, \times, \min, \max$ have their usual interpretations over $\mathbb{N}$. $f(n, \vec{n})$ is defined by *bounded recursion*, written BR, from g, h, k if $f(n,\vec{n}) \leq k(n,\vec{n})$ for all $n, \vec{n}$ and:

$$\begin{array}{rcl} f(0,\vec{n}) & = & g(\vec{n}) \\ f(n+1,\vec{n}) & = & h(n,\vec{n},f(n,\vec{n})) \end{array}$$

We write $\mathcal{E}^2$ for the function algebra $[0,1,+,\times,\min,\max,\pi^k_j;\mathsf{comp},\mathsf{BR}]$ over $\mathbb{N}$.

Let us write **FLINSPACE** for the class of functions on $\mathbb{N}$ computable in linear space (see, e.g., [5]). The following result is well-known:

▶ **Proposition 19** ([21]). $\mathcal{E}^2 = \mathbf{FLINSPACE}$.

For a list of arguments $\vec{x} = (x_1, \dots, x_k)$, let us write $|\vec{x}|$ for $(|x_1|, \dots, |x_k|)$.

▶ **Lemma 20.** *For $f(\vec{x}) \in u$C, there is a $l_f(\vec{n}) \in \mathcal{E}^2$ such that $|f(\vec{x})| = l_f(|\vec{x}|)$.*

Proof. We proceed by induction on the definition of f in $u\mathsf{C}$. For the initial functions we have: $|\varepsilon| = 0$, $|\mathsf{s}_0 x| = |x| + 1$, $|\mathsf{s}_1 x| = |x| + 1$, $|x \# y| = |x||y|$, $|\pi^k_j(x_1, \dots, x_n)| = |x_j|$, $|x \wedge y| = \min(|x|, |y|)$, and $|x \vee y| = \max(|x|, |y|)$.

If f is defined by composition, the result is immediate from composition in $\mathcal{E}^2$. Finally, if $f(x, \vec{x})$ is defined by $u\mathsf{BRN}$ from functions $g, h, k \in u\mathsf{C}$, as in (5), then we have,

$$\begin{array}{rcl} |f(\varepsilon, \vec{x})| & = & |g(\vec{x})| \\ |f(\mathsf{s}_i x, \vec{x})| & = & |h(1, x, \vec{x}, f(x, \vec{x}))| \end{array}$$

and we may define l_f by BR from l_g, l_h and l_k, by the inductive hypothesis. ◀

By appealing to the lengths of $\varepsilon, \mathsf{s}_1, \cdot, \#, \wedge, \vee, u\mathsf{BRN}$, we also have a converse result to Lemma 20 above, giving the following characterisation of the tally functions of $u\mathsf{C}$:

▶ **Theorem 21.** *Let $f : \mathbb{N}^k \to \mathbb{N}$. Then the binary string function $\underline{f(|\vec{x}|)}$ is in $u\mathsf{C}$ if and only if the natural number function $f(\vec{n})$ is in $\mathcal{E}^2$.*

Proof sketch. The left-right implication follows from Lemma 20 above, and the right-left implication follows by simulating $\mathcal{E}^2$-definitions with unary codings in $u\mathsf{C}$. ◀

Thanks to this result, we will rather work in $\mathcal{E}^2$ when reasoning about tally functions in $u\mathsf{C}$, relying on known facts about **FLINSPACE** (see, e.g., [5]).

In $u\mathsf{C}$, we may also use unary codings to "iterate" other functions. We write $f(\underline{\vec{n}}, \vec{y}) \in u\mathsf{C}$ if there is $f'(\vec{x}, \vec{y}) \in u\mathsf{C}$ such that $f'(\underline{\vec{n}}, \vec{y}) = f(\underline{\vec{n}}, \vec{y})$, for all $\vec{n} \in \mathbb{N}$.

▶ **Observation 22** (Length iteration). *$u\mathsf{C}$ is closed under the* bounded length iteration *operation: we may define $f(\underline{n}, \vec{x})$ from $g(\vec{x})$, $h(\underline{n}, \vec{x}, y)$ and $k(\underline{n}, \vec{x})$ as:*

$$\begin{array}{rcl} f(\underline{0}, \vec{x}) & := & g(\vec{x}) \\ f(\underline{n+1}, \vec{x}) & := & h(\underline{n}, \vec{x}, f(\underline{n}, \vec{x})) \end{array}$$

as long as $|f(\underline{n}, \vec{x})| \leq |k(\underline{n}, \vec{x})|$.

In fact, bounded length iteration is just a special case of $u\mathsf{BRN}$, and we will implicitly use this when iterating functions by length. This is crucial for deriving closure properties of $u\mathsf{C}$, as in the next subsection, and for showing that $u\mathsf{C} \supseteq$ **posFP** in Sect. 6.

▶ Remark (Some iterated functions). For $h(x, \vec{x}) \in u\mathsf{C}$, the following functions are in $u\mathsf{C}$:

$$\begin{array}{rclcrcl} \bigvee\limits_{j<|x|} h(j, \vec{x}) & := & h\left(\underline{|x|-1}, \vec{x}\right) \vee \cdots \vee h\left(\underline{0}, \vec{x}\right) & \qquad & \bigvee x & := & \bigvee\limits_{j<|x|} bit\,(j, x) \\ \bigwedge\limits_{j<|x|} h(j, \vec{x}) & := & h\left(\underline{|x|-1}, \vec{x}\right) \wedge \cdots \wedge h\left(\underline{0}, \vec{x}\right) & & \bigwedge x & := & \bigwedge\limits_{j<|x|} bit\,(j, x) \end{array}$$

$$\bigodot_{j<|x|} h(j, \vec{x}) \quad := \quad h\left(\underline{|x|-1}, \vec{x}\right) \cdot \cdots \cdot h\left(\underline{0}, \vec{x}\right)$$

Notice that, as for the definitions of $\bigvee x$ and $\bigwedge x$ above, we may use iterated operators with various limit formats, implicitly assuming that these are definable in $u\mathsf{C}$.

▶ **Example 23** (A program for sorting). Notice that the recurrence in (1), while an instance of BRN, is not an instance of $u\mathsf{BRN}$, since it is not uniform. However, we may give a positive definition by $u\mathsf{BRN}$ based, once again, on the recurrence (2):

$$\begin{array}{rcl} sort(\varepsilon) & = & \varepsilon \\ sort(\mathsf{s}_i x) & = & \bigodot\limits_{j<|x|} \left(bit(j+1, \mathsf{s}_1 sort(x)) \vee (i \wedge bit(j, \mathsf{s}_1 sort(x)))\right) \end{array}$$

5.2 $u\mathsf{C}$ is closed under simultaneous uBRN

To exemplify the robustness of the algebra $u\mathsf{C}$ it is natural to show closure under certain variants of recursion. While we do not explicitly use these results later, the technique should exemplify how other textbook-style results may be obtained for $u\mathsf{C}$. We also point out that the ideas herein are implicitly used in Sect. 6 where we inline a treatment of a restricted version of "course-of-values" recursion.

One of the difficulties in reasoning about $u\mathsf{C}$ is that it is not clear how to define appropriate (monotone) (de)pairing functions, which are usually necessary for such results. Instead, we rely on analogous results for $\mathcal{E}^2$, before "lifting" them to $u\mathsf{C}$, thanks to Thm. 21 and Prop. 2. We give a self-contained exposition for the benefit of the reader but, since **FLINSPACE** and algebras like $\mathcal{E}^2$ are well known, we will proceed swiftly; see, e.g., [5] for more details.

Notice that we have the following functions in $\mathcal{E}^2$,

$$n \dot{-} m := \max(n-m, 0) \qquad \text{and} \qquad \mathit{cond}_0(x,y,z) := \begin{cases} y & \text{if } x = 0 \\ z & \text{otherwise} \end{cases}$$

thanks to Thm. 21 and the fact that $\mathit{msp}(|x|, y)$ and $\mathit{cond}_\varepsilon$ are in $u\mathsf{C}$. Thus we may define,

$$\mathit{le}(m,n) \ := \begin{cases} 0 & \text{if } n \dot{-} m = 0 \\ 1 & \text{otherwise} \end{cases} \qquad \text{and} \qquad \left\lfloor \frac{n}{2} \right\rfloor \ := \sum_{i<n} \mathit{le}(2i+1, n)$$

by bounded recursion. This allows us to define in $\mathcal{E}^2$ a simple pairing function:

▶ **Proposition 24** (Pairing in $\mathcal{E}^2$)**.** *The following function is in $\mathcal{E}^2$:*

$$\langle n_0, n_1 \rangle := \left\lfloor \frac{(n_0+n_1)(n_0+n_1+1)}{2} \right\rfloor + n_0$$

We now show that we have the analogous *depairing* functions, due to the fact that bounded minimisation is available in **FLINSPACE**.

▶ **Lemma 25** (Bounded minimisation, [12])**.** *$\mathcal{E}^2$ is closed under* bounded minimisation*: if $f(n,\vec{n}) \in \mathcal{E}^2$ then so is the following function:*

$$\mathsf{s}(\mu m < n).(f(m,\vec{n}) = 0) \quad := \quad \begin{cases} m+1 & m < n \textit{ is least s.t. } f(m,\vec{n}) = 0 \\ 0 & f(m,\vec{n}) > 0 \textit{ for all } m < n \end{cases}$$

Proof. Appealing to BR, we have $\mathsf{s}(\mu m < 0).(f(m,\vec{n}) = 0) = 0$ and,

$$\begin{aligned} & \mathsf{s}(\mu m < n+1).(f(m,\vec{n}) = 0) \\ = & \begin{cases} n+1 & \text{if } \mathsf{s}(\mu m < n).(f(m,\vec{n}) = 0) = 0 \ , \ f(n,\vec{n}) = 0 \\ 0 & \text{if } \mathsf{s}(\mu m < n).(f(m,\vec{n}) = 0) = 0 \ , \ f(n,\vec{n}) \neq 0 \\ \mathsf{s}(\mu m < n).(f(m,\vec{n}) = 0) & \text{if } \mathsf{s}(\mu m < n).(f(m,\vec{n}) = 0) \neq 0 \end{cases} \end{aligned}$$

by two applications of the conditional cond_0. ◀

▶ **Proposition 26** (Depairing)**.** *For $i \in \{0,1\}$, the function β_i with $\beta_i(\langle n_0, n_1\rangle) = n_i$ is in $\mathcal{E}^2$.*

Proof. We have $\beta_0(n) = \mathsf{s}(\mu n_0 < n).(\mathsf{s}(\mu n_1 < n).(\langle n_0, n_1 \rangle = n) \neq 0) \ \dot{-} \ 1$, which is definable by bounded minimisation and appropriate conditionals.[7] $\beta_1(n)$ is defined analogously, by switching $\mathsf{s}(\mu n_0 < n)$ and $\mathsf{s}(\mu n_1 < n)$. ◀

[7] Notice that $\langle n_0, n_1 \rangle = n$ iff $\max(\langle n_0, n_1\rangle \dot{-} n, n \dot{-} \langle n_0, n_1 \rangle) = 0$.

Thanks to (de)pairing, we have the following (well-known) result:

▶ **Proposition 27.** *$\mathcal{E}^2$ is closed under* simultaneous bounded recursion*: we may define $f_1, \dots, f_p$ from $g_1, h_1, k_1 \dots, g_p, h_p, k_p$ if $f_j(n, \vec{n}) \leq k_j(n, \vec{n})$ for all $n, \vec{n}$, for $1 \leq j \leq p$, and:*

$$\begin{array}{rcl} f_j(0, \vec{n}) & = & g_j(\vec{n}) \\ f_j(n+1, \vec{n}) & = & h_j(n, \vec{n}, f_1(n, \vec{n}), \cdots, f_p(n, \vec{n})) \end{array}$$

This result, along with Lemma 20, allows us to show that $u\mathsf{C}$ is closed under the simultaneous form of $u\mathsf{BRN}$, by using *concatenation* instead of pairing:

▶ **Theorem 28.** *$u\mathsf{C}$ is closed under* simultaneous $u\mathsf{BRN}$*: we may define $f_1, \dots, f_p$ from $g_1, h_1, k_1 \dots, g_p, h_p, k_p$ if $|f_j(x, \vec{x})| \leq |k_j(x, \vec{x})|$ for all $x, \vec{x}$, for $1 \leq j \leq p$, and:*

$$\begin{array}{rcl} f_j(\varepsilon, \vec{x}) & = & g_j(\vec{x}) \\ f_j(\mathsf{s}_i x, \vec{x}) & = & h_j(i, x, \vec{x}, f_1(x, \vec{x}), \dots, f_p(x, \vec{x})) \end{array}$$

Proof sketch. For $1 \leq j \leq p$, we have g_j, h_j, k_j are in $u\mathsf{C}$, therefore by Lemma 20 there exist, in $\mathcal{E}^2$, functions l_{g_j}, l_{h_j} and l_{k_j} computing their output lengths in terms of their input lengths. Appealing to simultaneous bounded recursion (Prop. 27), we may define in the natural way functions $l_{f_j} \in \mathcal{E}^2$ such that $|f_j(x, \vec{x})| = l_{f_j}(|x|, |\vec{x}|)$ for all $x, \vec{x}$.

Now, using concatenation, we define the following function in $u\mathsf{C}$ by $u\mathsf{BRN}$,

$$\begin{array}{rcl} F(\varepsilon, \vec{x}) & = & g_1(\vec{x}) \cdot \cdots \cdot g_p(\vec{x}) \\ F(\mathsf{s}_i x, \vec{x}) & = & h_1(i, x, \vec{x}, \vec{F}(x, \vec{x})) \cdot \cdots \cdot h_p(i, x, \vec{x}, \vec{F}(x, \vec{x})), \end{array}$$

where $\vec{F} = (F_1, \dots F_p)$ and each $F_j(x, \vec{x})$ is $F(x, \vec{x})$ without its leftmost $l_{f_1}(|x|, |\vec{x}|) + \cdots + l_{f_{j-1}}(|x|, |\vec{x}|)$ and its rightmost $l_{f_{j+1}}(|x|, |\vec{x}|) + \cdots + l_{f_p}(|x|, |\vec{x}|)$ bits, i.e.,

$$F_j(x, \vec{x}) = msp\left(l_{f_{j+1}}(|x|, |\vec{x}|) + \cdots + l_{f_p}(|x|, |\vec{x}|), F(x, \vec{x})\right) \wedge \underline{l_{f_j}(|x|, |\vec{x}|)}$$

The bounding function is just the concatenation of all the $k_j(x, \vec{x})$, for $1 \leq j \leq p$. Now we may conclude by noticing that $f_j(x, \vec{x}) = F_j(x, \vec{x})$, for $1 \leq j \leq p$. ◀

6 uC contains posFP

We are now ready to present our proof of the converse to Thm. 17. For this we appeal to the characterisation (1) from Thm. 7 of **posFP** as Δ_0-uniform families of ¬-free circuits. Since Δ_0 formulae compute just the predicates of the linear-time hierarchy, the following result is not surprising, though we include it for completeness of the exposition:

▶ **Lemma 29** (Characteristic functions of Δ_0 sets)**.** *Let φ be a Δ_0-formula with free variables amongst $\vec{n}$. There is a function $f_\varphi(\vec{n}) \in \mathcal{E}^2$ such that:*

$$f_\varphi(\vec{n}) \quad = \quad \begin{cases} 0 & \mathbb{N} \not\models \varphi(\vec{n}) \\ 1 & \mathbb{N} \models \varphi(\vec{n}) \end{cases}$$

Proof. We already have functions for all terms (written s, t, etc.), i.e. polynomials, due to the definition of $\mathcal{E}^2$. We proceed by induction on the structure of φ, which we assume by De Morgan duality is written over the logical basis $\{\neg, \wedge, \forall\}$:

- For atomic formulae we use the length conditional to define appropriate functions:

$$f_{s<t}(\vec{n}) \ := \ \begin{cases} 1 & s \dot{-} (t+1) = 0 \\ 0 & \text{otherwise} \end{cases} \qquad f_{s=t}(\vec{n}) \ := \ \begin{cases} 1 & \max(s \dot{-} t, t \dot{-} s) = 0 \\ 0 & \text{otherwise} \end{cases}$$

- If φ is $\neg\psi$ then we define f_φ, using the conditional, as follows:

$$f_\varphi(\vec{n}) \;:=\; \begin{cases} 1 & f_\psi(\vec{n}) = 0 \\ 0 & \text{otherwise} \end{cases}$$

- If φ is $\psi \wedge \chi$ then we define f_φ as follows:

$$f_\varphi(\vec{n}) \;:=\; \min(f_\psi(\vec{n}), f_\chi(\vec{n}))$$

- If φ is $\forall n < t.\psi(n,\vec{n})$ then we define $f_\varphi(t,\vec{n})$, by BR, as follows:

$$\begin{aligned} f_\varphi(0,\vec{n}) \;&:=\; 1 \\ f_\varphi(n+1,\vec{n}) \;&:=\; \min\left(f_\psi(n,\vec{n}), f_\varphi(n,\vec{n})\right) \end{aligned}$$ ◀

Using this result, we may argue for the converse of Thm. 17.

▶ **Theorem 30.** $\mathbf{posFP} \subseteq u\mathsf{C}$.

Proof. Working with the characterisation (1) from Thm. 7 of **posFP**, we use Lemma 29 above to recover characteristic functions of sets specifying a $\neg$-free circuit family $C(\vec{n})$ in $\mathcal{E}^2$. Writing $N, D, E, I_1, \dots, I_k, O$ for the associated characteristic functions (in $\mathcal{E}^2$), we define an "evaluator" program in $u\mathsf{C}$, taking advantage of Thm. 21, that progressively evaluates the circuit as follows. Given inputs $\vec{x}$ of lengths $\vec{n}$, we will define a function $Val(\underline{n},\vec{x})$ that returns the concatenation of the outputs of the gates $< n$ in $C(\vec{n})$, by length iteration, cf. Obs. 22.

The base case of the iteration is simple, with $Val(\underline{0},\vec{x}) := \varepsilon$. For the inductive step we need to set up some intermediate functions. Suppressing the parameters $\vec{n}$, we define the function $\iota(\underline{n},\vec{x})$ returning the concatenation of input bits sent to the n^{th} gate:

$$\iota(\underline{n},\vec{x}) \;:=\; \bigodot_{m<|x_1|} \left(\underline{I_1(m,n)} \wedge bit(\underline{m},x_1)\right) \cdot \cdots \cdot \bigodot_{m<|x_k|} \left(\underline{I_k(m,n)} \wedge bit(\underline{m},x_k)\right)$$

Now we define the value $val(\underline{n},\vec{x})$ of the n^{th} gate in terms of $Val(\underline{n},\vec{x})$, appealing again to the iterated operators from Rmk. 5.1, and testing for the empty string:[8]

$$val(\underline{n},\vec{x}) \;:=\; \begin{cases} \bigwedge \iota(\underline{n},\vec{x}) \;\wedge\; \bigwedge\limits_{m<n} \left(\underline{(1 \dot{-} E(m,n))} \vee bit(\underline{m},\, Val(\vec{n},\vec{x}))\right) & \text{if } \underline{D(n)} = \underline{0} \\ \bigvee \iota(\underline{n},\vec{x}) \;\vee\; \bigvee\limits_{m<n} \left(\quad \underline{E(m,n)} \quad\; \wedge bit(\underline{m},\, Val(\vec{n},\vec{x}))\right) & \text{if } \underline{D(n)} = \underline{1} \end{cases}$$

Finally we may define $Val\,(\underline{n+1},\vec{x}) := val(\underline{n},\vec{x}) \cdot Val(\underline{n},\vec{x})$. At this point we may define the output $C(\vec{x})$ of the circuit as $\bigodot\limits_{m<N} \left(\underline{O(m)} \wedge bit(\underline{m},\, Val(\underline{N},\vec{x}))\right)$. ◀

7 A characterisation based on safe recursion

In [3] Bellantoni and Cook give an *implicit* function algebra for **FP**, not mentioning any explicit bounds, following seminal work by Leivant, [16, 17], who first gave a *logical* implicit characterisation of **FP**. In this section we give another function algebra for **posFP** in the style of Bellantoni and Cook's, using "safe recursion". Our argument follows closely the structure of the original argument in [3]; it is necessary only to verify that those results go through once an appropriate uniformity constraint is imposed. We write normal-safe functions as usual: $f(\vec{x};\vec{y})$ where $\vec{x}$ are the *normal* inputs and $\vec{y}$ are the *safe* inputs.

[8] Formally, here we follow the usual convention that $\bigvee \varepsilon = 0$ and $\bigwedge \varepsilon = 1$.

▶ **Definition 31** (Function algebra $u\mathsf{B}$)**.** We say that f is defined by *safe composition*, written scomp, from functions $g, \vec{r}, \vec{s}$ if: $f(\vec{x};\vec{y}) = g(\vec{r}(\vec{x};);\vec{s}(\vec{x};\vec{y}))$. We say that f is defined by *uniform safe recursion on notation* ($u\mathsf{SRN}$) from functions g and h if:

$$\begin{array}{rcl} f(\varepsilon,\vec{x};\vec{y}) & = & g(\varepsilon,\vec{x};\vec{y}) \\ f(\mathsf{s}_i x,\vec{x};\vec{y}) & = & h(x,\vec{x};i,\vec{y},f(x,\vec{x};\vec{y})) \end{array}$$

We define $u\mathsf{B} := [\varepsilon, \mathsf{s}_0^{;1}, \mathsf{s}_1^{;1}, \pi_j^{l;k}, \wedge^{;2}, \vee^{;2}, p^{;1}, cond_\varepsilon^{;3}; \mathsf{scomp}, u\mathsf{SRN}]$. Here, superscripts indicate the arity of the function, which we often omit. We will show that the normal part of $u\mathsf{B}$ computes precisely **posFP**, following the same argument structure as [3].

▶ **Lemma 32** (Bounding lemma)**.** *For all $f \in$ uB, there is a polynomial $b_f(\vec{m},\vec{n})$ (with natural coefficients) such that, for all $\vec{x},\vec{y}$, $|f(\vec{x};\vec{y})| \leq b_f(|\vec{x}|,|\vec{y}|)$.*

Proof idea. We show by that for $f \in u\mathsf{B}$, by induction on its definition, there exists a polynomial $q_f(\vec{n})$ such that, for all $\vec{x},\vec{y}$, $|f(\vec{x};\vec{y})| \leq q_f(|\vec{x}|) + \max_j(|y_j|)$. (This is just a special case of the same property for B from [3].) ◀

▶ **Proposition 33.** *If $f(\vec{x};\vec{y}) \in u\mathsf{B}$, then we have $f(\vec{x},\vec{y}) \in u\mathsf{C}$.*

Proof sketch. We proceed by induction on the definition of f; the only interesting case is when f is defined by $u\mathsf{SRN}$. In this case we define f analogously to $u\mathsf{BRN}$, taking the bounding function to be $\underline{b_f(|\vec{x}|,|\vec{y}|)}$, where b_f is obtained from Lemma 32 above. ◀

Therefore we have that $u\mathsf{B}$ is contained in $u\mathsf{C}$, and consequently in **posFP**. In order to establish the other inclusion we slightly reformulate the function algebra $u\mathsf{C}$. We write $u\mathsf{C}' := [\varepsilon, s_0, s_1, \pi_j^n, \wedge, \vee; \mathsf{comp}, u\mathsf{BRN}']$, where $u\mathsf{BRN}'$ is defined as $u\mathsf{BRN}$ but with the bounding polynomial $k \in [\varepsilon, s_1, \pi_j^n, \cdot, \#; \mathsf{comp}]$. It is clear that $u\mathsf{C}$ is contained in $u\mathsf{C}'$; namely the function $\#$ can easily be defined (as in, e.g., the proof of Prop. 35 later). We will prove that $u\mathsf{C}'$ is contained in $u\mathsf{B}$.

▶ **Lemma 34.** *For all $f \in u\mathsf{C}'$ there is a polynomial $p_f(n)$ and some $f'(w;\vec{x}) \in u\mathsf{B}$ such that, for all $\vec{x}, w$, $(|w| \geq p_f(|\vec{x}|) \Rightarrow f(\vec{x}) = f'(w;\vec{x}))$.*

Proof sketch. The proof is similar to the proof of the analogous statement for **FP** given in [3], with routine adaptations to deal with uniformity. We proceed by induction on the definition of f in $u\mathsf{C}'$, with the interesting case being when f is defined by $u\mathsf{BRN}'$, say from functions g, h and k. Let g', p_g, h' and p_h be the appropriate functions and polynomials obtained by the inductive hypothesis. We would like to define $f' \in u\mathsf{B}$ and a polynomial p_f such that, for all $w, x, \vec{x}$, whenever $|w| \geq p_f(|x|,|\vec{x}|)$ one has $f(x,\vec{x}) = f'(w;x,\vec{x})$. The problem is that in $u\mathsf{B}$, due to the normal-safe constraints, one cannot define f' directly by recursion on x. Therefore we introduce in $u\mathsf{B}$ some auxiliary functions. Define,

$$\begin{array}{rcl} msp(|\varepsilon|;y) & := & y \\ msp(|\mathsf{s}_i x|;y) & := & p(;msp(|x|;y)) \end{array} \qquad \begin{array}{rcl} msp(|x|,y;) & := & msp(|x|;y) \\ X(z,w;x) & := & msp(|msp(|z|,w;)|;x) \\ I(z,w;x) & := & X(\mathsf{s}_1 z,w;x) \wedge 1 \end{array}$$

by $u\mathsf{SRN}$ and by safe composition. The function X is used to "simulate" the recursion over x, with x in a safe input position. Now, by $u\mathsf{SRN}$, we define $F(\varepsilon,w;x,\vec{x}) := \varepsilon$ and,

$$\begin{array}{l} F(\mathsf{s}_i z,w;x,\vec{x}) \\ := \begin{cases} g'(w;\vec{x}) & \text{if } X(\mathsf{s}_1 z,w;x) = \varepsilon \\ h'(w;I(z,w;x),X(z,w;x),\vec{x},F(z,w;x,\vec{x})) & \text{otherwise} \end{cases} \end{array} \tag{6}$$

using a length conditional, cf. Prop. 16. From here we set $f'(w;x) := F(w,w;x,\vec{x})$ and also,

$$p_f(|x|,|\vec{x}|) \quad := \quad p_h(1,|x|,|\vec{x}|,b_f(|x|,|\vec{x}|)) + p_g(|\vec{x}|) + |x| + 1,$$

where b_f is a polynomial bounding the length of the outputs of f (which exists since $f \in u\mathsf{C}'$.)

Given $x, \vec{x}$, take w such that $|w| \geq p_f(|x|,|\vec{x}|)$. We will prove, by subinduction on $|u|$, that, if $|w| - |x| \leq |u| \leq |w|$, then $F(u,w;x,\vec{x}) = f(X(u,w;x),\vec{x})$. Since $X(w,w;x) = x$, we thus obtain that $f'(w;x,\vec{x}) = F(w,w;x,\vec{x}) = f(x,\vec{x})$, as required.

Let us take an arbitrary u such that $|w| - |x| \leq |u| \leq |w|$. Note that $|w| - |x| \geq 1$, and thus we may write $u = \mathsf{s}_i z$ for some z. We have two cases:

- If $|\mathsf{s}_i z| = |w| - |x|$ then $X(\mathsf{s}_i z, w; x) = \varepsilon$, and so $F(\mathsf{s}_i z, w; x, \vec{x}) = g'(w;\vec{x}) = g(\vec{x}) = f(\varepsilon,\vec{x}) = f(X(\mathsf{s}_i z, w; x), \vec{x})$.
- If $|\mathsf{s}_i z| > |w| - |x|$ then $X(\mathsf{s}_i z, w; x) \neq \varepsilon$ and so:

$$\begin{array}{rcll} F(\mathsf{s}_i z, w; x, \vec{x}) & = & h'(w; I(z,w;x), X(z,w;x), \vec{x}, F(z,w;x,\vec{x})) & \text{by (6)} \\ & = & h(I(z,w;x), X(z,w;x), \vec{x}, F(z,w;x,\vec{x})) & \text{by inductive hypothesis} \\ & = & h(I(z,w;x), X(z,w;x), \vec{x}, f(X(z,w;x),\vec{x})) & \text{by subinductive hypothesis} \\ & = & f(X(\mathsf{s}_i z, w; x), \vec{x}) & \text{by definition of } f. \end{array}$$

◀

▶ **Proposition 35.** *If $f(\vec{x})$ in $u\mathsf{C}$, then we have $f(\vec{x};) \in u\mathsf{B}$.*

Proof. For f in $u\mathsf{C}$, recalling that $u\mathsf{C} \subseteq u\mathsf{C}'$, take $f' \in u\mathsf{B}$ and a polynomial p_f given by Lemma 34 above. It suffices to prove that there exists $r \in u\mathsf{B}$ such that $|r(\vec{x};)| \geq p_f(|\vec{x}|)$, for all $\vec{x}$, whence we have $f'(r(\vec{x};);\vec{x}) = f(\vec{x})$ as required, cf. Lemma 34 above. For this we simply notice that the usual definitions of polynomial growth rate functions, e.g. from [3], can be conducted in unary, using only uniform recursion. Namely, define $\oplus$ and $\otimes$ in $u\mathsf{B}$ as follows, by $u\mathsf{SRN}$,

$$\begin{array}{rcl} \oplus(\varepsilon;y) & := & y \\ \oplus(\mathsf{s}_i x; y) & := & \mathsf{s}_1(\oplus(x;y)) \end{array} \qquad \begin{array}{rcl} \otimes(\varepsilon,y;) & := & \varepsilon \\ \otimes(\mathsf{s}_i x, y;) & := & \oplus(y;\otimes(x;y)) \end{array}$$

so that $|\oplus(x;y)| = |x| + |y|$ and $|\otimes(x,y;)| = |x| \times |y|$. By safe composition we may also write $\oplus(x,y;)$ in $u\mathsf{B}$, yielding an appropriate function $r(\vec{x};) \in u\mathsf{B}$. ◀

As a consequence of Props. 33 and 35 in this section, and Thms. 17 and 30 earlier, we summarise the contributions of this work in the following characterisation:

▶ **Theorem 36.** $u\mathsf{B} = u\mathsf{C} = \mathbf{posFP}$.

8 Conclusions

In this work we observed that characterisations of "positive" polynomial-time computation in [14] are similarly robust in the functional setting. We gave a function algebra $u\mathsf{C}$ for **posFP** by *uniformising* the recursion scheme in Cobham's characterisation for **FP**, and gave a characterisation based on safe recursion too. We also observed that the tally functions of **posFP** are precisely the unary encodings of **FLINSPACE** functions on $\mathbb{N}$.

$u\mathsf{C}$ has a natural generalisation for arbitrary ordered alphabets, not just $\{0,1\}$. This is similarly the case for the circuit families and machine model we presented in Sect. 2. We believe these, again, induce the same class of functions, and can even be embedded monotonically into $\{0,1\}$, thanks to appropriate variants of $u\mathsf{BRN}$ in $u\mathsf{C}$, e.g. Thm. 28.

Unlike for non-monotone functions, there is an interesting divergence between the monotone functions on binary words and those on the integers. Viewing the latter as *finite sets*, characterised by their binary representaion, we see that the notion of monotonicity induced by $\subseteq$ is actually more restrictive than the one studied here on binary words. For example, natural numbers of different lengths may be compared, and the *bit* function is no longer monotone. In fact, a natural way to characterise such functions would be to *further* uniformise recursion schemes, by also relating the base case to the inductive step, e.g.:

$$\begin{array}{rcl} f(0,\vec{x}) & = & h(0,0,\vec{x},0) \\ f(\mathsf{s}_i x,\vec{x}) & = & h(i,x,\vec{x},f(x,\vec{x})) \end{array}$$

Adapting such recursion schemes to provide a "natural" formulation of the positive polynomial-time predicates and functions on $\mathbb{N}$ is the subject of ongoing work.

Finally, this work serves as a stepping stone towards providing *logical theories* whose provably recursive functions correspond to natural monotone complexity classes. *Witnessing theorems* for logical theories typically compile to function algebras on the computation side, and in particular it would be interesting to see if existing theories for monotone *proof complexity* from [8] appropriately characterise positive complexity classes. We aim to explore this direction in future work.

References

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

2 David A. Mix Barrington, Neil Immerman, and Howard Straubing. On Uniformity within NC^1. *J. Comput. Syst. Sci.*, 41(3):274–306, 1990. doi:10.1016/0022-0000(90)90022-D.

3 Stephen Bellantoni and Stephen A. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992. doi:10.1007/BF01201998.

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

5 Peter Clote and Evangelos Kranakis. *Boolean Functions and Computation Models*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2002. doi:10.1007/978-3-662-04943-3.

6 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.

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

8 Anupam Das. From positive and intuitionistic bounded arithmetic to monotone proof complexity. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 126–135, 2016. doi:10.1145/2933575.2934570.

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

10 Michelangelo Grigni. *Structure in monotone complexity*. PhD thesis, Duke University, 1991.

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

12 Andrzej Grzegorczyk. *Some classes of recursive functions*. Instytut Matematyczny Polskiej Akademi Nauk, 1953.

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

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

15 Clemens Lautemann, Thomas Schwentick, and Iain A. Stewart. Positive versions of polynomial time. *Inf. Comput.*, 147(2):145–170, 1998. doi:10.1006/inco.1998.2742.

16 Daniel Leivant. A foundational delineation of computational feasiblity. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 2–11, 1991. doi:10.1109/LICS.1991.151625.

17 Daniel Leivant. A foundational delineation of poly-time. *Inf. Comput.*, 110(2):391–420, 1994. doi:10.1006/inco.1994.1038.

18 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.

19 Christos H. Papadimitriou. *Computational complexity.* Academic Internet Publ., 2007.

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

21 Robert W. Ritchie. Classes of predictably computable functions. *Journal of Symbolic Logic*, 28(3):252–253, 1963.

22 Walter L. Ruzzo. On uniform circuit complexity. *J. Comput. Syst. Sci.*, 22(3):365–383, 1981. doi:10.1016/0022-0000(81)90038-6.

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

24 Celia Wrathall. Complete sets and the polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):23–33, 1976. doi:10.1016/0304-3975(76)90062-1.