Monotonicity Constraints in Characterisations of PSPACE

Amir M. Ben-Amram [a] Bruno Loff [b,*] Isabel Oitavem [c]

[a] *Tel-Aviv Academic College*

[b] *CWI, Amsterdam*

[c] *FCT-UNL and CMAF-UL, Lisbon*

Abstract

A celebrated contribution of Bellantoni and Cook was a function algebra to capture FPTIME. This algebra uses recursion on notation. Later, Oitavem showed that including primitive recursion, an algebra is obtained which captures FPSPACE. The main results of this paper concern variants of the latter algebra. First, we show that iteration can replace primitive recursion. Then, we consider the results of imposing a monotonicity constraint on the primitive recursion or iteration. We find that in the case of iteration, the power of the algebra shrinks to FPTIME. More interestingly, with primitive recursion, we obtain a new implicit characterisation of the polynomial hierarchy (FPH).

The idea to consider these monotonicity constraints arose from the results on write-once tapes for Turing machines. We review this background and also note a new machine characterisation of Δ_2^P, that similarly to our function algebras, arises by combining monotonicity constraints with a known characterisation of PSPACE.

Key words: complexity classes, implicit characterisations, recursion schemes

1 Introduction

1.1 Motivation and overview

It is an open problem whether P = PSPACE. An algorithm working in polynomial space is allowed to reuse space, and if we look at known algorithms

* Corresponding author. Address is *CWI, Science Park Amsterdam, Kruislaan 413, NL-1098 SJ Amsterdam.*

Email addresses: benamram.amir@gmail.com (Amir M. Ben-Amram), bruno.loff@gmail.com (Bruno Loff), oitavem@fct.unl.pt (Isabel Oitavem).

for PSPACE-complete problems, they always seem to rely heavily on this possibility. Our intuition then indicates that this is a crucial point concerning the problem P versus PSPACE. A rigorous formulation of this intuition is the known fact that a "write-once" Turing machine, given polynomial space, decides exactly P (see Section 2). A write-once machine is one which is not allowed to erase (or rewrite) cells which have been previously written on.

Since the write-once restriction can, in some sense, be seen as a monotonicity constraint on the contents of the storage, this suggests investigating how sensitive characterisations of PSPACE are to "monotonicity constraints".

The core of this paper explores, in particular, the interplay between recursion and iteration. [1] In Section 4, we take as starting point the recursion-theoretic characterisation of FPSPACE given in [Oitavem, 1997], and we show that substituting predicative primitive iteration for predicative primitive recursion also leads to FPSPACE (note that here we are concerned with function classes rather than classes of decision problems. All characterisations by means of recursion schemes will, naturally, be of function classes. However, in sections that discuss characterisations by Turing machines, we use the ordinary language classes).

In Section 5, we show that imposing a monotonicity constraint on the above recursion and iteration operators leads, in the case of primitive iteration, to FPTIME, and, in the case of primitive recursion, to the polynomial hierarchy FPH. We form a hierarchy based on the nesting-level of the restricted primitive recursion operator, and this provides a new implicit characterisation of all levels of the polynomial hierarchy. This result resembles previous work by Bellantoni [1995], and we discuss the connection. To conclude, we present a novel machine characterisation of Δ_2.

1.2 Preliminaries

We use $\mathbb{W}$ to denote the set of binary words $\{0,1\}^*$. For $z \in \mathbb{W}$, we use z' to denote its numerical successor, defined recursively in the following way:

$$\varepsilon' = 0 \qquad (w0)' = w1 \qquad (w1)' = w'0$$

We use z^- to denote the numerical predecessor (the word such that $(z^-)' = z$, for $z \neq \varepsilon$). The numerical order over $\mathbb{W}$ is denoted by $\leqslant$.

Given $x \in \mathbb{W}$, $|x|$ is the length of x, and for $x_1, \ldots, x_k \in \mathbb{W}$, we set $|\vec{x}| = |x_1, \ldots, x_k| = \sum_{i=1}^{k} |x_i|$.

A function $f : \mathbb{W}^n \to \mathbb{W}$, for any $n > 0$, is *polynomially bounded* if there is a polynomial p such that $|f(\vec{x})| \leqslant p(|\vec{x}|)$ for all $\vec{x}$.

[1] Which is a time-honoured subject of sub-recursion theory [e.g. Gladstone, 1971].

2 Polynomial Write-Once Space

The class of **polynomial time problems**, P, is the class of subsets of $\mathbb{W}$ decidable by a Turing machine in polynomial time. It is well-known that classification by polynomial time or space is robust with respect to the number of tapes or tape heads of the Turing machine.

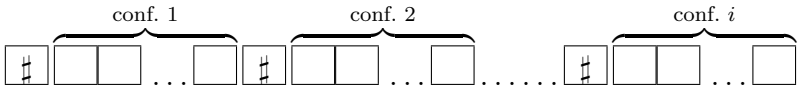
Definition 1 *A **write-once Turing machine** is a Turing machine equipped with an input (read-only) tape and a write-once work-tape. The work-tape has one reading-head (that can only read), and one writing-head. The writing-head moves exclusively to the right, cannot be moved without writing (though it can stay put), and cannot write a blank*[2] *. The class of **polynomial write-once space problems**,* PWOS*, is the class of sets decidable by write-once Turing machine in polynomial space.*

We prove in this section that polynomial write-once space is equivalent to polynomial time on Turing machines. This result is essentially due to Irani, Naor and Rubinfeld [1992], who proved the same connection for a certain type of random-access machine. We provide the proof for write-once Turing machines for completeness, and also because it motivated our work in the subsequent sections. In addition, near the end of this paper, we will use this model as a basis for a new machine characterisation of Δ_2.

Proposition 2 $\mathrm{P} \subseteq \mathrm{PWOS}$.

Proof. We begin by showing that write-once Turing machines (of the dual-head variant) with polynomial space complexity can decide any set in P. Let $A \in \mathrm{P}$ and $\mathcal{M}$ a single-tape, single-head Turing machine with tape alphabet Γ that decides if $x \in A$ in time n^k, where $n = |x|$. We assume that $\mathcal{M}$'s tape is semi-infinite (does not extend to the left of the initial position). We simulate $\mathcal{M}$ in polynomial write-once space. Our write-once Turing machine, $\mathcal{M}^\dagger$, will have $2|\Gamma| + 1$ work symbols. Two symbols α and $\dot{\alpha}$ for each symbol $\alpha \in \Gamma$ and one additional *separator* symbol $\sharp$.

The principle of the simulation is that each successive tape-configuration of $\mathcal{M}$ (the tape contents and head position) is written out in full on $\mathcal{M}^\dagger$'s tape. Successive configurations are separated by the separator symbol, $\sharp$. A dotted symbol ($\dot{\alpha}$) is used to mark the head position in each configuration.



[2] We can also consider a single-head variant where a single head can be used in two "modes," either scanning a portion of the tape in a read-only manner, or writing in a previously-blank cell. Unlike the two-head machine, this model can scan a blank portion of the tape without writing anything. Despite this subtle difference, the results presented here apply to both.

The machine creates the initial configuration by writing $\sharp$ and then copying the input while marking the first symbol. At the end of this stage, the reading-head scans the $\sharp$ while the writing-head scans a blank cell following the configuration. Throughout the simulation, the simulator maintains in its control the current control-state of $\mathcal{M}$.

Next, each step of $\mathcal{M}$ is simulated in the following way:

(1) a separator ($\sharp$) is written by the writing-head.

(2) Moving both heads simultaneously to the right, the tape contents of the last configuration are copied to a new configuration, until the reading-head finds a marked symbol $\dot{\alpha}$. When starting step i of this copying phase, the writing-head is ready to write the $i-1$st symbol of the new configuration, while the reading-head scans the ith symbol of the last configuration (the value of the $i-1$st symbol of the last configuration is maintained in the machine's control). Next, the writing-head writes the $i-1$st symbol of the last configuration to the new configuration; the ith symbol, seen by the reading-head, is recorded in the machine's control and the head moves to the $i+1$st symbol.

(3) Once the position of $\mathcal{M}$'s head has been reached, $\mathcal{M}^\dagger$ simulates the appropriate transition of $\mathcal{M}$, copying, if necessary (i.e., on a right move) also the symbol following the head's position. So, for instance, if the vicinity of the reading-head is

$$\ldots \boxed{\sigma}\boxed{\dot{\alpha}}\boxed{\alpha}\boxed{\pi} \ldots$$

and the transition of $\mathcal{M}$ replaces α with γ and moves to the right, then the reading-head advances up to the π, while the writing-head writes

$$\ldots \boxed{\sigma}\boxed{\gamma}\boxed{\dot{\alpha}}$$

(and is now located on the blank cell following the $\dot{\alpha}$). Had $\mathcal{M}$ moved to the left, the writing-head would have written $\dot{\sigma}\gamma\alpha$ instead.

If $\mathcal{M}$ accepts or rejects, then $\mathcal{M}^\dagger$ does the same. Otherwise, $\mathcal{M}$'s new control state is recorded in $\mathcal{M}^\dagger$'s.

(4) Moving both heads simultaneously to the right again, the rest of the configuration is copied, until the reading-head finds the separator.

This simulation is obviously done in polynomial space (approximately the product of $\mathcal{M}$'s time and space), and the restrictions of the write-once model are obeyed [3]. □

Proposition 3 PWOS $\subseteq$ P

Proof. A write-once Turing machine $\mathcal{M}$ with s states and working in space n^k can perform at most $s \cdot n^{k+1}$ computational steps without writing to the tape and without looping. Since it is write-once, it can only write on the tape at most n^k times. So, without looping, the machine can perform at most $s \cdot n^{2k+1}$ steps. Thus, PWOS $\subseteq$ P. □

[3] If the single-head model were to be used, the ability to leave a gap, that is, skip over a blank cell, could be exploited to keep track of the position copied from.

Theorem 4 P *is the class of problems which can be solved in polynomial write-once space.* □

So, we have the following nice intuition about time versus space:

> Limiting time is the same as limiting the number of re-writes, *or*, limiting time rather than space is the same as loosing your eraser.

A way of explaining the difference between a read/write memory and a write-once memory is that the contents of the latter change monotonically. Theorem 4 shows that imposing this monotonicity constraint on a formulation of PSPACE yields a characterisation of P. This motivates us to consider monotonicity in conjunction with a different way of characterising space-bounded computation: namely by recursion schemes. This is the subject of the next three sections.

3 Function Algebras

A function algebra is a characterisation of a set of functions by the inductive closure, under some operators, of an initial set of functions. This concept is frequently used in recursion theory, and more recently to obtain characterisations of complexity classes [Clote, 1999].

Definition 5 *Let $\mathscr{F}$ be a class of functions, let $\mathbb{F} \subseteq \mathscr{F}$ be a set of such functions, and let $\mathbb{O} \subseteq \cup_{k\in\mathbb{N}}\{O : \mathscr{F}^k \to \mathscr{F}\}$ be a set of operators. The* ***inductive closure*** *of $\mathbb{F}$ under $\mathbb{O}$, written $\mathbb{A} = [\mathbb{F};\mathbb{O}]$, is the smallest set containing $\mathbb{F}$, such that if $f_1, \ldots, f_k \in \mathbb{A}$ are in the domain of the k-ary $O \in \mathbb{O}$, then $O(f_1, \ldots, f_k) \in \mathbb{A}$.*

We also refer to $[\mathbb{F};\mathbb{O}]$ as a ***function algebra****; the carrier of the algebra is $\mathbb{A}$ and its operations are $\mathbb{O}$.* [4]

In this work, $\mathbb{F}$ and $\mathbb{O}$ are always finite.

We make liberal use of the square brackets, e.g., if f, g are functions, $\mathbb{F}$ is a class of functions, and $O_1, \ldots, O_k$ are operators, then we set

$$[f, g, \mathbb{F}; O_1, \ldots, O_n] = [\{f, g\} \cup \mathbb{F}; \{O_1, \ldots, O_n\}].$$

Definition 6 *Let $\mathbb{A} = [\mathbb{F};\mathbb{O}]$ be a function algebra, for a set $\mathbb{F} = \{f_1, f_2, \ldots\}$ of functions and a set $\mathbb{O} = \{O_1, O_2, \ldots\}$ of operators. We write $\mathbb{D}_\mathbb{A}$ to stand for the set of* ***descriptions*** *for $\mathbb{A}$, where a description is a term in a term algebra containing*

(1) an atom $\mathtt{fun}_i$ for each $f_i \in \mathbb{F}$, which is said to ***describe*** *f_i;*
(2) for each operator $O_i \in \mathbb{O}$ of arity k, the terms $\mathtt{Op}_i(d_1, \ldots, d_k)$, for any descriptions $d_1, \ldots, d_k$ which describe $g_1, \ldots, g_k$ in the domain of O_i. This

[4] A rigorous notation for the algebra would be $(\mathbb{A}, \mathbb{O}) = ([\mathbb{F};\mathbb{O}], \mathbb{O})$; but the distinction will be ignored to simplify notation. No confusion should arise.

term is said to describe $O_i(g_1, \ldots, g_k)$.

Descriptions have natural syntactic measures of complexity, and these can induce complexity measures for the function algebra. A frequently considered measure is the nesting-depth of a certain operator (or a set of operators, but this generality is not necessary in this paper).

Definition 7 *Let* $\mathbb{A} = [\mathbb{F}; \mathbb{O}]$ *be a function algebra, with* $\mathbb{O} = \{O_1, O_2, \ldots\}$, *and let* $O_r \in \mathbb{O}$. *The* ***rank*** *of a description* $d \in \mathbb{D}_{\mathbb{A}}$ *with respect to* O_r, $\mathrm{rk}(d)$, *is inductively defined as*

(1) $\mathrm{rk}(\mathtt{fun}_i) = 0$,
(2) if $i \neq r$, *then* $\mathrm{rk}(\mathtt{Op}_i(d_1, \ldots, d_k)) = \max(\mathrm{rk}(d_1), \ldots, \mathrm{rk}(d_k))$, *and*
(3) $\mathrm{rk}(\mathtt{Op}_r(d_1, \ldots, d_k)) = \max(\mathrm{rk}(d_1), \ldots, \mathrm{rk}(d_k)) + 1$.

The ***rank of a function*** $f \in \mathbb{A}$ *with respect to* O_r, $\mathrm{rk}(f)$, *is given by*

$$\mathrm{rk}(f) = \min\{\mathrm{rk}(d) : d \textit{ describes } f\}.$$

Definition 8 *Let* $\mathbb{A} = [\mathbb{F}; \mathbb{O}]$ *be a function algebra, and let* $O_r \in \mathbb{O}$. *The* ***rank hierarchy*** *in* $\mathbb{A}$ *with respect to* O_r *is the sequence* H_n *defined by*

$$H_n = \{f \in \mathbb{A} : \mathrm{rk}(f) \leqslant n\}.$$

The next proposition explains the rank hierarchy in terms of the inductive closure: the next level of the hierarchy is obtained by allowing one further application of O_r and closing under the other operators.

Proposition 9 *Let* $\mathbb{A} = [\mathbb{F}; \mathbb{O}]$ *be a function algebra, let* $O_r \in \mathbb{O}$, *and set* $\mathcal{V} = \mathbb{O} - \{O_r\}$. *The rank hierarchy in* $\mathbb{A}$ *with respect to* O_r *can be inductively defined by:*

(1) $H_0 = [\mathbb{F}; \mathcal{V}]$,
(2) $I_n = H_n \cup \{O_r(f_1, \ldots, f_k) : (f_1, \ldots, f_k) \in (H_n)^k \cap Dom(O_r)\}$, *and*
(3) $H_{n+1} = [I_n; \mathcal{V}]$.

We will skip the proof, which is a simple induction. □

The advantage of expressing the hierarchy as above (in contrast with Definition 8) is the convenience of proving statements about the hierarchy by induction on the rank.

3.1 Two-sorted function algebras for polynomial time

Take $\mathscr{F}$ to be the class of functions with two argument sorts, as in [Bellantoni and Cook, 1992]: Each function $f \in \mathscr{F}$, where $f : \mathbb{W}^n \to \mathbb{W}$, is associated with a number k, $0 \leqslant k \leqslant n$, so that the first k arguments of f are of the *normal* sort, and the remaining arguments are of the *safe* sort; we indicate the division by a semicolon, writing $F(x_1, \ldots, x_k; x_{k+1}, \ldots, x_n)$ or just $F(\vec{x}; \vec{y})$. Such functions will be called in this paper "two-sorted functions."

We equate classes of two-sorted functions with classes of ordinary (one-sorted) functions in the following sense:

Definition 10 *Let $\mathbb{F}$ be a class of functions and $\mathcal{F}$ be a class of two-sorted functions.*

(1) We write $\mathbb{F} \subseteq \mathcal{F}$ when every $f \in \mathbb{F}$ is also in $\mathcal{F}$, for some division of its arguments into sorts.
(2) We write $\mathcal{F} \subseteq \mathbb{F}$ when every function $f(\vec{x};\vec{y}) \in \mathcal{F}$ is in $\mathbb{F}$ when we ignore the sorts (i.e., $f(\vec{x},\vec{y}) \in \mathbb{F}$).
(3) We write $\mathbb{F} \simeq \mathcal{F}$ when $\mathbb{F} \subseteq \mathcal{F}$ and $\mathcal{F} \subseteq \mathbb{F}$.

Notice that for any given $\mathcal{F}$ there is a unique $\mathbb{F}$ satisfying $\mathcal{F} \simeq \mathbb{F}$ (which is just $\mathcal{F}$ with no sorts). So, for two classes $\mathcal{F}, \mathcal{F}'$ of two-sorted functions, we will write $\mathcal{F} \simeq \mathcal{F}'$ if $\mathcal{F} \simeq \mathbb{F}$ and $\mathcal{F}' \simeq \mathbb{F}$ for the same $\mathbb{F}$. This induces an equivalence relation.

Consider the set $\mathscr{B}$ of basic functions, containing (i–viii) defined below:

i. **(source)**
ε (a zero-ary function)
ii. **(projections)**
$\pi_i^{k,n}(x_1,\ldots,x_k;x_{k+1},\ldots,x_{k+n}) = x_i$, for each $1 \leqslant i \leqslant k+n$
iii. **(normal binary successors)**
$S_i(x;) = xi,\ i \in \{0,1\}$
iv. **(bounded safe binary successors)**
$$S_i(z;x) = \begin{cases} xi \text{ if } |x| < |z| \\ x \quad \text{otherwise} \end{cases}$$
v. **(binary predecessor)**
$P(;\varepsilon) = \varepsilon, P(;xi) = x$
vi. **(numerical predecessor)**
$p(;\varepsilon) = \varepsilon, p(;x') = x$
vii. **(conditional)**
$Q(;\varepsilon,y,z_0,z_1) = y, Q(;xi,y,z_0,z_1) = z_i$
viii. **(tally product)**
$\times(x,y;) = 1^{|x|\times|y|}$.

Now consider the following operators:

ix. **(predicative composition)** Given $g,\vec{r},\vec{s}$, their predicative composition, $f = \mathrm{C}(g,\vec{r},\vec{s})$ is defined by
$f(\vec{x};\vec{y}) = g(\vec{r}(\vec{x};);\vec{s}(\vec{x};\vec{y}))$.
x. **(predicative recursion on notation)** Given g,h_0,h_1, the predicative recursion on notation scheme defines a function $f = \mathrm{R}(g,h_0,h_1)$ by
$f(\varepsilon,\vec{x};\vec{y}) = g(\vec{x};\vec{y})$,
$f(zi,\vec{x};\vec{y}) = h_i(z,\vec{x};\vec{y},f(z,\vec{x};\vec{y}))$.
xi. **(predicative primitive recursion)** Given g,h, the predicative primitive recursion scheme defines a function $f = \mathcal{R}(g,h)$ by

$f(\varepsilon, \vec{x}; \vec{y}) = g(\vec{x}; \vec{y})$,
$f(z', \vec{x}; \vec{y}) = h(z, \vec{x}; \vec{y}, f(z, \vec{x}; \vec{y}))$.

Note that the function's value is treated as sort-less. Note also that if $g(\vec{x}; \vec{y})$, then the function f defined by $f(\vec{x}, \vec{y};) = g(\vec{x}; \vec{y})$ can be obtained by composition and projections. Thus, when discussing any class $\mathcal{F}$ that includes at least $\mathscr{B}$ and closed under predicative composition (such as all classes discussed in this paper), if one knows that some unsorted function h has a counterpart in $\mathcal{F}$, it is certain that $h(\vec{x};)$ is in $\mathcal{F}$.

Looking at the two recursion schemes (x and xi), note that only normal arguments can be used as induction variables, while the result of a recursive call is put in a safe position. Thus, informally, it is not possible to effect a "double recursion" leading to exponential growth in the length of the words.

It *is* possible to compute any polynomial in the following sense:

Lemma 11 *For any polynomial $r : \mathbb{N} \to \mathbb{N}$, the two-sorted function $(\vec{x};) \mapsto 1^{r(|\vec{x}|)}$ is in $[\mathscr{B}; \mathrm{C}, \mathrm{R}]$.* □

Based on [Bellantoni and Cook, 1992] and [Oitavem, 1997], we have:

Theorem 12 *(i)* $\mathrm{FPTIME} \simeq [\mathscr{B}; \mathrm{C}, \mathrm{R}]$. *(ii)* $\mathrm{FPSPACE} \simeq [\mathscr{B}; \mathrm{C}, \mathrm{R}, \mathcal{R}]$. □

Where FPSPACE denotes the class of polynomially bounded functions computable in polynomial space.

Given a class $\mathbb{F}$ of polynomially-bounded functions, use $\mathrm{FPTIME}(\mathbb{F})$ to denote the set of functions Cook-reducible to some $f \in \mathbb{F}$, that is, computable in polynomial time given an f oracle. We may extend the result from [Bellantoni and Cook, 1992] in the following way.

Theorem 13 *Let $\mathbb{F}$ be a class of polynomially-bounded functions, and $\mathcal{F}$ a class of two-sorted functions. Suppose that*

(I) for all $f \in \mathbb{F}$ there exists $F(w; \vec{x}) \in \mathcal{F}$ and a polynomial p such that

$$\forall \vec{x} \forall w \; |w| \geqslant p(|\vec{x}|) \implies f(\vec{x}) = F(w; \vec{x}).$$

Then $\mathrm{FPTIME}(\mathbb{F}) \subseteq [\mathscr{B} \cup \mathcal{F}; \mathrm{C}, \mathrm{R}]$.

Less formally, this states that if we take some class $\mathbb{F}$, then the closure of $\mathbb{F}$ under Cook reductions is obtained by adding the basic functions $\mathscr{B}$ to $\mathbb{F}$, and closing under predicative composition and predicative recursion on notation — so Bellantoni and Cook's characterisation of FPTIME *relativises*, in some sense. For proving Theorem 13, and other theorems in this paper, we need the following observation, which gives the connection of FPTIME to $[\mathscr{B}; \mathrm{C}, \mathrm{R}]$ in more detail:

Theorem 14 (Bellantoni and Cook [1992]) *Item (I) of Theorem 13 above holds for $\mathbb{F} = \mathrm{FPTIME}$, $\mathcal{F} = [\mathscr{B}; \mathrm{C}, \mathrm{R}]$.*

Proof of Theorem 13. Let $\mathscr{P}$ denote the inductive closure $[\mathscr{B} \cup \mathcal{F}; \mathrm{C}, \mathrm{R}]$.

Let $\mathcal{M}$ be a polynomial-time single-tape oracle Turing machine, that computes

a function f in polynomial time by means of a function oracle $g \in \mathbb{F}$. Such a machine has a work tape, an oracle tape and an output tape. Let us clarify the usage of the oracle: the machine has a special *query state*. Whenever it enters this state, the contents of the oracle tape are interpreted as a query $\vec{x}$ and instantaneously replaced by the result $g(\vec{x})$.

Using (I), let $G \in \mathcal{F}$ be such that $G(w; \vec{x}) = g(\vec{x})$ for all $\vec{x}$ and $|w| \geqslant p_G(|\vec{x}|)$.

Suppose that the machine has tape alphabet Γ and states Q. Let b be fixed so that $2^b > |\Gamma \cup Q|$. We encode a configuration of $\mathcal{M}$ as a word $q\#x\#y\#u\#v\#w$, where q is the current state, x, y represent portions of the work-tape before and after the head position (the head is scanning the first symbol of y), and similarly u, v for the oracle tape; w is the contents of the output tape (which is write-only). Each symbol of this word is encoded in binary, using b bits. Since $\mathcal{M}$ is polynomial-time, and g is polynomially bounded, we can fix a polynomial p_2 such that every reachable configuration on input $\vec{x}$ can be encoded (with padding if necessary) in exactly $p_c(|\vec{x}|)$ bits.

This encoding can be manipulated by the following functions, all in $\mathscr{P}$ (this is easy to justify by Theorems 12(i) and 14, since the functions are polynomial-time computable).

(1) A function *initial* such that $\mathit{initial}(\vec{x};)$ encodes the initial configuration of $\mathcal{M}$ for the input $\vec{x}$;
(2) A function *step* and a polynomial p_{st} such that, given a word c encoding some configuration of $\mathcal{M}$, a word a, and any word $|w| \geqslant p_{st}(|c, a|)$, $\mathit{step}(w; c, a)$ will encode the next configuration of $\mathcal{M}$. If c is in the query state, then a is interpreted as the word written on the answer tape. If c is in a final state, then $\mathit{step}(w; c, a) = c$.
(3) A function *query* and a polynomial p_{qr} such that, given a word c encoding some configuration of $\mathcal{M}$, and any word $|w| \geqslant p_{qr}(|c|)$, $\mathit{query}(w; c)$ is the word in the query tape of $\mathcal{M}$ for this configuration.
(4) A function *out* such that, given a word c encoding some configuration of $\mathcal{M}$, $\mathit{out}(c;)$ is the word in the output tape of $\mathcal{M}$ for this configuration.

Now define, by predicative recursion on notation:

$$F(\varepsilon, w, \vec{x};) = \mathit{initial}(\vec{x};)$$

$$F(zi, w, \vec{x};) = \mathit{step}(w;\ F(z, w, \vec{x};),\ G(w; \mathit{query}(w; F(z, w, \vec{x};))))$$

Let $c = F(z, w, \vec{x};)$ be a valid configuration. Then

(1) From our assumption on the coding of configurations, $|c| = p_c(|\vec{x}|)$;
(2) we may also assume that, if $|w| \geqslant p_G(|c|) + p_{qr}(|c|)$, $|G(w; \mathit{query}(w; c))| \leqslant |c|$, since both the query and its answer must fit in the configuration;
(3) and thus, whenever $|w| \geqslant p_{st}(2|c|)$, $\mathit{step}(w; c, G(w; \mathit{query}(w; c)))$ will correctly encode the configuration of $\mathcal{M}$ that follows configuration c.

It follows that for a sufficiently large polynomial $p(n)$, and $|w| \geqslant p(|\vec{x}|)$, $F(z, w, \vec{x};)$ encodes the configuration of $\mathcal{M}$ with oracle g on input $\vec{x}$ after

$|z|$ steps. And so, if r is a polynomial such that $r(|\vec{x}|)$ bounds the running time of $\mathcal{M}$ on input $\vec{x}$, we have

$$f(\vec{x}) = out(F(1^{r(|\vec{x}|)}, 1^{p(|\vec{x}|)}, \vec{x};);).$$

Since $\mathcal{M}$ and g are arbitrary, we conclude that FPTIME($\mathbb{F}$) $\subseteq \mathscr{P}$. □

4 Two-Sorted Algebras for FPSPACE

We will next consider a modification of the algebra $[\mathscr{B}; \mathrm{C}, \mathrm{R}, \mathcal{R}]$ by replacing the primitive recursion scheme with an *iteration* construct. Informally, iteration differs from recursion in that the recursive procedure can not use the recursion variable in computations. In imperative programming terms, this is akin to having a "repeat n times" control structure instead of "for $z = 1, \ldots, n$ do". Formally, we introduce the following operator:

xii. **(predicative iteration)** Given g, h, the predicative iteration scheme defines a function $f = \mathcal{I}(g, h)$ by
$f(\varepsilon, \vec{x}; \vec{y}) = g(\vec{x}; \vec{y})$,
$f(z', \vec{x}; \vec{y}) = h(\vec{x}; \vec{y}, f(z, \vec{x}; \vec{y}))$.

Definition 15 *(1) $\mathcal{A}$ denotes the function algebra $[\mathscr{B}; \mathrm{C}, \mathrm{R}, \mathcal{R}]$;*
(2) $\mathcal{B}$ denotes the function algebra $[\mathscr{B}; \mathrm{C}, \mathrm{R}, \mathcal{I}]$.

We already know (Theorem 12) that $\mathcal{A} \simeq$ FPSPACE. We show that this is also the case for $\mathcal{B}$.

Theorem 16 $\mathcal{B} \simeq$ FPSPACE

Proof. We know that $\mathcal{A} \simeq$ FPSPACE. Now clearly $\mathcal{B} \subseteq \mathcal{A}$, since it is trivial to define predicative iteration using predicative primitive recursion, and thus $\mathcal{B} \subseteq$ FPSPACE. To see that FPSPACE $\subseteq \mathcal{B}$, we show that one may simulate any polynomial-space Turing computation. Choose any Turing machine $\mathcal{M}$ computing some FPSPACE function f. As in the proof of Theorem 13, we may fix a method of encoding configurations of $\mathcal{M}$ so that the following hold. First, any configuration in the computation of $\mathcal{M}$ over some input $\vec{x}$ is encoded by a binary word of size exactly $p(|\vec{x}|)$, for some fixed polynomial p. Secondly, there exist in $\mathcal{B}$

(1) A function *initial* such that $initial(\vec{x};)$ encodes the initial configuration of $\mathcal{M}$ for the input $\vec{x}$.
(2) (by Theorem 14) A function *step* and a polynomial q such that, given a word c encoding some configuration of $\mathcal{M}$, and any word $|w| \geqslant q(|c|)$, $step(w; c)$ will encode the next configuration of $\mathcal{M}$.
(3) A function *out* such that, given a word c encoding some configuration of $\mathcal{M}$, $out(c;)$ is the word in the output tape of $\mathcal{M}$ for this configuration.

So we may see that

$$F(\varepsilon, w, \vec{x};) = \mathit{initial}(\vec{x};) \qquad F(z', w, \vec{x};) = \mathit{step}(w; F(z, w, \vec{x};))$$

defines a function F by predicative iteration, and thus $F \in \mathcal{B}$. But it is easy to show by induction that, provided that $|w| \geqslant q(p(|x|))$, $F(z, w, \vec{x};)$ encodes the configuration of $\mathcal{M}$ after m steps of computation over the input $\vec{x}$, where m is the number of numeric predecessors of z. So if r is a polynomial such that $2^{r(|\vec{x}|)}$ bounds the number of steps required for $\mathcal{M}$ to finish its computation, then we define

$$f'(\vec{x};) = \mathit{out}(F(1^{r(|\vec{x}|)}, 1^{q(p(|\vec{x}|))}, \vec{x};);),$$

we always have $f(\vec{x}) = f'(\vec{x};)$, and thus FPSPACE $\subseteq \mathcal{B}$. □

5 Restrictions of Two-Sorted Algebras for FPSPACE

Consider the following partial order over $\mathbb{W}$.

Definition 17 *For $w, v \in \mathbb{W}$, We write $w \preceq v$ if $|w| < |v|$, or $|w| = |v|$ and $\forall i(v_i \leqslant w_i)$. We write $w \prec v$ if $w \preceq v$ but $w \neq v$.*

Then $(\mathbb{W}, \preceq)$ is a partial order; e.g. $0011 \preceq 0111$ and $0011 \preceq 1011 \preceq 1111$, while 0011 and 0110 are incomparable.

Lemma 18 *Let $w^{(1)} \preceq w^{(2)} \preceq \cdots \preceq w^{(\ell)} \preceq v$. The number of distinct words in the sequence is at most $\frac{1}{2}(n+1)(n+2)$, where $n = |v|$.*

Proof. The number of distinct words of any given size k in a $\preceq$-ordered chain is at most $k+1$ (because we can switch a 0 to a 1 but not vice versa). So an upper bound on the number of distinct words in the given chain is:

$$\sum_{k=0}^{n}(k+1) = \frac{1}{2}(n+1)(n+2) \qquad \square$$

In fact, given any size n, there is a sequence with exactly $\frac{1}{2}(n+1)(n+2)$ distinct elements; e.g., for $n = 3$,

$$\varepsilon, 0, 1, 00, 01, 11, 000, 001, 011, 111.$$

Definition 19 *A two-sorted function h, with at least one safe argument, is called **monotone** whenever $z \preceq h(\vec{x}; \vec{y}, z)$ for all $z \in \mathbb{W}$.*

Definition 20 *We define the following operators over two-sorted functions:*

xiii. **(restricted predicative primitive recursion)** The restricted predicative primitive recursion operator, $\tilde{\mathcal{R}}$, is defined as $\tilde{\mathcal{R}}(g, h) = \mathcal{R}(g, h)$ if h is monotone, and undefined otherwise.

xiv. **(restricted predicative iteration)** The restricted predicative iteration operator $\tilde{\mathcal{I}}$, is defined as $\tilde{\mathcal{I}}(g, h) = \mathcal{I}(g, h)$ if h is monotone, and undefined otherwise.

We will study the consequences of replacing $\mathcal{R}$ and $\mathcal{I}$ by their respective restricted versions. The reader may find the resulting classes suspicious from a computability point of view, since it is undecidable whether a given description defines a monotone function, and therefore, whether the application of an operator is valid. In programming language terms, one cannot decide whether a given description is a well-formed program. This motivates us to incorporate the monotonicity into the definition of the recursion operator, as shown next.

Definition 21 *Let h be a two-sorted function with at least one safe argument. Its **monotone section** is the function*

$$h^{\mathrm{m}}(\vec{x};\vec{y},z) = \begin{cases} h(\vec{x};\vec{y},z) & \text{if } z \preceq h(\vec{x};\vec{y},z), \\ z & \text{otherwise.} \end{cases}$$

Clearly, h^{m} is always monotone.

Definition 22 ***(monotone recursion and iteration schemes)*** *Given g, h, the predicative monotone primitive recursion scheme is defined by $\mathcal{R}^{\mathrm{m}}(g,h) = \mathcal{R}(g,h^{\mathrm{m}})$. The predicative monotone iteration scheme is $\mathcal{I}^{\mathrm{m}}(g,h) = \mathcal{I}(g,h^{\mathrm{m}})$.*

The above operators coincide with the restricted ones if h is monotone, but are defined for any h and, clearly, in an effective manner.

Definition 23 *We define the following function classes:*

$\mathrm{m}\mathcal{A}$	*("monotone $\mathcal{A}$")*	*is*	$[\mathscr{B}; \mathrm{C}, \mathrm{R}, \mathcal{R}^{\mathrm{m}}]$
$\mathrm{m}\mathcal{B}$	*("monotone $\mathcal{B}$")*	*is*	$[\mathscr{B}; \mathrm{C}, \mathrm{R}, \mathcal{I}^{\mathrm{m}}]$
$\mathrm{r}\mathcal{A}$	*("restricted $\mathcal{A}$")*	*is*	$[\mathscr{B}; \mathrm{C}, \mathrm{R}, \tilde{\mathcal{R}}]$, *and*
$\mathrm{r}\mathcal{B}$	*("restricted $\mathcal{B}$")*	*is*	$[\mathscr{B}; \mathrm{C}, \mathrm{R}, \tilde{\mathcal{I}}]$.

Since the restricted operators coincide with the monotone ones over their domain, we immediately have $\mathrm{r}\mathcal{A} \subseteq \mathrm{m}\mathcal{A}$ and $\mathrm{r}\mathcal{B} \subseteq \mathrm{m}\mathcal{B}$. As a consequence of our complexity-class characterisations, we will be able to show that $\mathrm{m}\mathcal{A} \simeq \mathrm{r}\mathcal{A}$ and $\mathrm{m}\mathcal{B} \simeq \mathrm{r}\mathcal{B}$.

Before embarking on the study of these classes, we mention a useful lemma from [Oitavem, 1997].

Lemma 24 *If $f(\vec{x};\vec{y})$ is in $\mathcal{A}$, then there exists a polynomial p_f such that*

$$|f(\vec{x};\vec{y})| \leqslant \max\{p_f(|\vec{x}|), \max_i |y_i|\}.$$

Since $\mathcal{B}, \mathrm{r}\mathcal{A}, \mathrm{r}\mathcal{B} \subseteq \mathcal{A}$, the lemma also holds for these classes. It is not hard to verify (though this requires perusing the proof) that it also holds for $\mathrm{m}\mathcal{A}$ and $\mathrm{m}\mathcal{B}$.

5.1 The strength of restricted primitive iteration

The next theorem tells us, in essence, that FPTIME is closed under restricted and monotone predicative primitive iteration.

Theorem 25 $\mathrm{m}\mathcal{B} \simeq \mathrm{r}\mathcal{B} \simeq \mathrm{FPTIME}$

Proof. It is clear that $\mathrm{FPTIME} \subseteq \mathrm{r}\mathcal{B}$, since $\mathrm{r}\mathcal{B}$ includes $[\mathscr{B}; \mathrm{C}, \mathrm{R}]$. Recall that $\mathrm{r}\mathcal{B} \subseteq \mathrm{m}\mathcal{B}$. We next show by induction on the structure of $\mathrm{m}\mathcal{B}$ that $\mathrm{m}\mathcal{B} \subseteq \mathrm{FPTIME}$. The functions in $\mathscr{B}$ are trivially in FPTIME. The inductive steps for the operators of predicative composition and predicative recursion on notation were shown in [Bellantoni and Cook, 1992]. Now, suppose that $g(\vec{x}; \vec{y}), h(\vec{x}; \vec{y}, w) \in \mathrm{r}\mathcal{B}$ are both polynomial-time computable, and let $f = \mathcal{I}^{\mathrm{m}}(g, h) = \mathcal{I}(g, h^{\mathrm{m}})$. We will prove that f is polynomial-time computable, completing the inductive step for monotone iteration.

Note that from the assumption $h \in \mathrm{FPTIME}$ it easily follows that $h^{\mathrm{m}} \in \mathrm{FPTIME}$. Choose, by Lemma 24, a polynomial p_f such that

$$|f(z, \vec{x}; \vec{y})| \leqslant \max\{p_f(|z, \vec{x}|), \max_i |y_i|\}.$$

For brevity, let $k = \max\{p_f(|z, \vec{x}|), \max_i |y_i|\}$. Define the sequence

$$s_\varepsilon = f(\varepsilon, \vec{x}; \vec{y}) = g(\vec{x}; \vec{y}),$$

$$s_{z'} = f(z', \vec{x}; \vec{y}) = h^{\mathrm{m}}(\vec{x}; \vec{y}, s_z).$$

Then this sequence forms a chain

$$s_\varepsilon \preceq s_0 \preceq \ldots \preceq s_n \preceq \ldots \preceq 1^k,$$

and by Lemma 18, this sequence contains as most $\frac{1}{2}(k+1)(k+2) \in O(k^2)$ different elements. But notice that $s_{v'}$ only depends on s_v, and so if at some point $s_v = s_{v'}$, then $s_z = s_v$ for all $z \geqslant v$. Thus, the $O(k^2)$ different elements in the chain must all be in its beginning, and we may obtain s_z simply by calculating $s_\varepsilon, s_0, s_1, s_{00}, \ldots$ until we stop when some $s_v = s_{v'}$. Every calculation is polynomial-time in $|\vec{x}, \vec{y}, z|$ since $g, h^{\mathrm{m}} \in \mathrm{FPTIME}$ and $|s_i| \leqslant k$ is polynomial in $|z, \vec{x}, \vec{y}|$. □

5.2 The strength of restricted primitive recursion

While imposing the monotonicity restriction on the iteration construct collapsed the function class down to FPTIME, we will show that restricted primitive recursion yields a class presumably between FPTIME and FPSPACE: the polynomial hierarchy. Let us recall some definitions:

Definition 26 *We define* $\Sigma_0 = \Pi_0 = \Delta_0 = \mathrm{P}$, $\Sigma_{i+1} = \mathrm{NP}(\Sigma_i)$, $\Pi_{i+1} = \text{co-NP}(\Sigma_i)$, *and* $\Delta_{i+1} = \mathrm{P}(\Sigma_i)$. *We let* $PH = \cup_i \Delta_i = \cup_i \Sigma_i$.

Corresponding function classes are $\Box_i = \mathrm{FPTIME}(\Delta_i) = \mathrm{FPTIME}(\Sigma_{i-1})$*, and* $\mathrm{FPH} = \cup_i \Box_i = \mathrm{FPTIME}(\mathrm{PH})$.

In many texts, these class names are adorned with a P superscript (e.g., Σ_i^P) for distinction from the arithmetic hierarchy; since this paper only concerns the polynomial hierarchy, we omit the superscript.

If R is an n-ary relation over $\mathbb{W}$, $\chi_R(\vec{x};)$ denotes its characteristic function, with all arguments normal. That is, $\chi_R(\vec{x};) = 1 \Leftrightarrow R(\vec{x})$ and otherwise $\chi_R(\vec{x};) = 0$.

We next recall the characterisation of PH by polynomially-bounded quantifiers.

Definition 27 *Let R be a $(1+n)$-ary relation, q be some polynomial. Define the n-ary relation $\exists_q y R$ by*

$$\vec{x} \in \exists_q y R \iff (\exists y : |y| \leqslant q(|\vec{x}|))\ (y, \vec{x}) \in R.$$

Similarly, define $\forall_q y R$ by

$$\vec{x} \in \forall_q y R \iff (\forall y : |y| \leqslant q(|\vec{x}|))\ (y, \vec{x}) \in R.$$

Theorem 28 (Stockmeyer [1976], Wrathall [1976]) *$L \subseteq \mathbb{W}$ is in Σ_n if and only if there is a $(1+n)$-ary relation $R \in \mathrm{P}$ and polynomials q_i such that*

$$L = \exists_{q_1} y_1 \forall_{q_2} y_2 \ldots Q_n y_n R$$

where the ith quantifier Q_i is $\forall_{q_i}$ if i is even, and $\exists_{q_i}$ if i is odd.

It is well-known that $\mathrm{FPTIME} \subseteq \mathrm{FPH} \subseteq \mathrm{FPSPACE}$, but it remains open whether these inclusions are proper. In this section we show that restricting the primitive recursion by monotonicity changes the function class $\mathcal{A}$ from FPSPACE to FPH. Moreover, we will show that the rank hierarchy in $\mathrm{r}\mathcal{A}$ corresponds precisely to the $\Box_n$ hierarchy.

Definition 29 $\mathrm{r}\mathcal{A}_n$ *denotes the n-th level of the rank hierarchy with respect to $\tilde{\mathcal{R}}$ within $\mathrm{r}\mathcal{A}$; specifically, following Proposition 9, we define:*

(1) $\mathrm{r}\mathcal{A}_0 = [\mathscr{B}; \mathrm{C}, \mathrm{R}]$,
(2) $\mathrm{r}\mathcal{A}'_{n+1} = \mathrm{r}\mathcal{A}_n \cup \{\mathcal{R}(g,h) : g, h \in \mathrm{r}\mathcal{A}_n$ *and h is monotone*$\}$, *and*
(3) $\mathrm{r}\mathcal{A}_{n+1} = [\mathrm{r}\mathcal{A}'_{n+1}; \mathrm{C}, \mathrm{R}]$.

Classes $\mathrm{m}\mathcal{A}_n$ *(and* $\mathrm{m}\mathcal{A}'_n$*) are similarly defined with respect to* $\mathcal{R}^{\mathrm{m}}$.

Theorem 30 *For all n, $\Box_{n+1} \subseteq \mathrm{r}\mathcal{A}_n$.*

Proof. We will show, by induction on n, that

(I) for all $R \in \Sigma_n$ there exists $F(w; \vec{x}) \in \mathrm{r}\mathcal{A}_n$ and a polynomial p s.t.

$$\forall \vec{x} \forall w\ |w| \geqslant p(|\vec{x}|) \Rightarrow \chi_R(\vec{x}) = F(w; \vec{x}).$$

Since $\Box_{n+1} = \mathrm{FPTIME}(\Sigma_n)$, Theorem 30 will follow from Theorem 13.

For $n = 0$ this is Theorem 14. For the induction step, assume that (I) holds for every $R \in \Sigma_n$. By using the conditional and composition, we can *negate* $F(w;\vec{x})$ into $Q(;F(w;\vec{x}),1,1,0)$, and we then find that (I) also holds for Π_n.

Now take an arbitrary predicate $P = \exists_q yR \in \Sigma_{n+1}$, where $R \in \Pi_n$. I.e.,

$$\vec{x} \in \exists_q yR \iff (\exists y : |y| \leqslant q(|\vec{x}|))\ (y,\vec{x}) \in R.$$

Make the standard assumption that $(y,\vec{x}) \notin R$ whenever $|y| > q(|\vec{x}|)$. Since (I) holds for Π_n, let $F \in \mathrm{r}\mathcal{A}_n$ obey $F(w;y,\vec{x}) = \chi_R(y,\vec{x})$ for any $|w| \geqslant p_F(|y,\vec{x}|)$. Using the conditional and predicative composition, we can assume that $F(w;y,\vec{x}) \in \{0,1\}$ for every input.

Define f by restricted predicative primitive recursion in the following way:

$$f(\varepsilon,w;\vec{x}) = \varepsilon$$

$$\begin{aligned} f(z',w;\vec{x}) &= Q(;\ \ p(;f(z,w;\vec{x})),\ \ F(w;z,\vec{x}),\ \ f(z,w;\vec{x}),\ \ f(z,w;\vec{x})) \\ &= \begin{cases} F(w;z,\vec{x}) & \text{if } f(z,w,\vec{x};) \in \{\varepsilon,0\} \\ f(z,w;\vec{x}) & \text{otherwise.} \end{cases} \end{aligned}$$

Note that since $F(w;y,\vec{x}) \in \{0,1\}$, then f is monotone. It is obtained by composition of functions in $\mathrm{r}\mathcal{A}_n$: the conditional Q and predecessor p are in $\mathscr{B}$, and F is in $\mathrm{r}\mathcal{A}_n$. Thus $f \in \mathrm{r}\mathcal{A}'_n$. It is easy to show by induction that, for $|w| \geqslant p_F(|z,\vec{x}|)$,

$$f(z,w;\vec{x}) = \begin{cases} 1 & \text{if } (\exists y < z)\ R(y,\vec{x}) \\ 0 & \text{otherwise.} \end{cases}$$

Thus $G(z;\vec{x}) = f(z,1^{p_F(2|z|)};\vec{x})$, is the characteristic of P, when $|z| \geqslant |\vec{x}|$ and $|z| \geqslant q(|\vec{x}|)+1$. G is in $\mathrm{r}\mathcal{A}_{n+1}$, as we intended. □

Theorem 31 *For all n, $\mathrm{m}\mathcal{A}_n \subseteq \Box_{n+1}$.*

Proof. We have $\mathrm{m}\mathcal{A}_0 = [\mathscr{B};\mathrm{C},\mathrm{R}] \simeq \mathrm{FPTIME} = \Box_1$. We next assume that $\mathrm{m}\mathcal{A}_{n-1} \subseteq \Box_n$ and prove that $\mathrm{m}\mathcal{A}'_n \subseteq \Box_{n+1}$. To this end, we let $g,h \in \Box_n$ and prove that $f = \mathcal{R}^{\mathrm{m}}(g,h)$ is in $\Box_{n+1}$.

By assumption, g,h can be computed in polynomial time with an oracle A in Σ_{n-1}. Clearly, so can h^{m}.

The equality $f = \mathcal{R}^{\mathrm{m}}(g,h)$ means that $f(z,\vec{x};\vec{y}) = s_z$, where s_z is obtained from the sequence

$$s_\varepsilon = g(\vec{x};\vec{y}) \qquad\qquad s_{i'} = h^{\mathrm{m}}(i,\vec{x};\vec{y},s_i)$$

which is monotone, namely

$$s_\varepsilon \preceq s_0 \preceq s_1 \preceq s_{00} \preceq s_{01} \preceq \ldots \preceq s_i \preceq \ldots \preceq s_z.$$

The size of each element in this sequence is polynomially bounded in $|z,\vec{x},\vec{y}|$, by Lemma 24. By Lemma 18, we know that s_i can only change a number

of times quadratic in $|s_z|$. So if we take $\mathfrak{I} = \{i_0, \ldots, i_k\}$ to be the sequence containing ε, z, and the indices i between ε and z at which $s_i \neq s_{i^-}$, then k is bounded by $O(|s_z|^2)$, and hence polynomial in $|z, \vec{x}, \vec{y}|$.

Our goal is to compute $f(z, \vec{x}; \vec{y})$ by first computing $s_{i_0} = s_\varepsilon = g(\vec{x}; \vec{y})$ and taking $i_0 = \varepsilon$, and then obtaining all the i_j's and s_{i_j}'s in sequence until we reach $s_{i_k} = s_z = f(z, \vec{x}; \vec{y})$. We will show is that given i_j and s_{i_j}, it is possible to obtain i_{j+1} and $s_{i_{j+1}}$ in polynomial time $p(|z, \vec{x}, \vec{y}|)$ with access to a Σ_n oracle. Thus the whole sequence can be generated in $kp(|z, \vec{x}, \vec{y}|))$ time, which is polynomial in $|z, \vec{x}, \vec{y}|$.

Let B be the following oracle:

$$(\vec{x}, \vec{y}, a, b, s) \in B \iff (\exists i : a \leqslant i \leqslant b)\ h^{\mathrm{m}}(i, \vec{x}; \vec{y}, s) \neq s$$

Since $h^{\mathrm{m}} \in \square_n$, the predicate "$h^{\mathrm{m}}(i, \vec{x}; \vec{y}, s) \neq s$" is in Δ_n, and thus $B \in \Sigma_n$, so also $A \oplus B \in \Sigma_n$, where $\oplus$ is the join operation ($x0 \in A \oplus B \Leftrightarrow x \in A$; and $x1 \in A \oplus B \Leftrightarrow x \in B$).

Given access to the oracle $A \oplus B$, and given i_j, s_{i_j}, we may use binary search to find the least i where $i_j < i \leqslant z$ and $h^{\mathrm{m}}(i, \vec{x}; \vec{y}, s_{i_j}) \neq s_{i_j}$. This is exactly i_{j+1}, and the binary search is done in time $\mathrm{O}(|z|^2)$ on an oracle Turing machine. Finally, by hypothesis, $s_{i_{j+1}} = h^{\mathrm{m}}(i_{j+1}, \vec{x}; \vec{y}, s_{i_j})$ can be computed with oracle $A \oplus B$ in time polynomial in $|i_{j+1}, \vec{x}, \vec{y}, s_{i_j}|$, which is again polynomial in $|z, \vec{x}, \vec{y}|$.

So $\mathrm{m}\mathcal{A}'_{n+1} \subseteq \square_{n+1}$, and by closure properties of the latter, $\mathrm{m}\mathcal{A}_{n+1} \subseteq \square_{n+1}$. $\square$

Combining the two inclusions, and the trivial $\mathrm{r}\mathcal{A}_n \subseteq \mathrm{m}\mathcal{A}_n$, we obtain

Theorem 32 *For all n, $\mathrm{m}\mathcal{A}_n \simeq \mathrm{r}\mathcal{A}_n \simeq \square_{n+1}$; hence, $\mathrm{m}\mathcal{A} \simeq \mathrm{r}\mathcal{A} \simeq \mathrm{FPH}$.*

5.3 Bellantoni's characterisation of $\square_n$

Bellantoni [1995] characterises the polynomial hierarchy by recursion schemes, in a way which resembles our results. Instead of restricted primitive recursion, Bellantoni includes the minimisation operator:

$$f(\vec{x}; \vec{y}) = \begin{cases} (\mu b.\ h(\vec{x}; \vec{y}, b) \bmod 2 = 0)1, & \text{if there is such a } b \\ 0 & \text{otherwise} \end{cases}$$

Bellantoni shows that the behaviour of the minimisation operator over functions of the class considered is such that the search can be bounded to values of size polynomial in the size of the input. Provided with such a bound, it is easy to simulate such a minimisation using restricted primitive recursion. Conversely, the algorithm in the proof of Theorem 31, together with Bellantoni's results, show that the inverse simulation is also possible.

6 Concluding remarks

We began by the observation that polynomial write-once space is equivalent to polynomial time, and observed that the essential feature of write-once memory is monotonicity. We then applied monotonicity constraints to two implicit characterisations of PSPACE: one based on predicative iteration and the other on predicative primitive recursion. We obtained different results, however, which may at first sight seem surprising. In retrospect, we would say that the restricted primitive recursion can still make an essential use of the recursion variable (in contrast with restricted iteration); note that the recursion variable changes (as a word) in a non-monotone way. Thus, while restricted iteration does not transcend FPTIME, restricted primitive recursion has the power to capture the polynomial hierarchy.

We would like to know whether recursion on notation is an indispensable operator in the definitions of $\mathcal{A}, \mathcal{B}, \mathrm{r}\mathcal{A}$ and $\mathrm{r}\mathcal{B}$. E.g., to which extent do we need to change our set $\mathscr{B}$ of basic functions into $\mathscr{B}'$ so that $\mathrm{r}\mathcal{A} = [\mathscr{B}; \mathrm{C}, \mathrm{R}, \tilde{\mathcal{R}}] = [\mathscr{B}'; \mathrm{C}, \tilde{\mathcal{R}}]$?

We conclude the paper by proposing a new machine characterisation of Δ_2.

6.1 Machine characterisation of Δ_2

After the characterisation of $\Box_{n+1}$, we asked whether we could go back to machines, and obtain a new machine characterisation of (levels of) the polynomial hierarchy.

We have not found an elegant machine characterisation of Δ_{n+1} in general. However, the case of Δ_2 seems sufficiently interesting. It fits the overall program of our research, since it results of plugging write-once tapes into the safe-storage machines of Cai and Furst [1991] which characterise PSPACE. We now sketch this construction.

Definition 33 *A write-once safe-storage machine is a Turing machine with a read-only* input tape*, two write-once tapes called the* work tape *and the* safe-storage tape*, and one* clock *(a read-only tape that is used as a counter).*

A computation of such a safe-storage machine begins with the input x written on the input tape, and with empty write-once tapes and clock. The computation continues either until the machine accepts or rejects, or until it enters a special tick state. Upon entering this state the work tape is erased, the clock is incremented to the numerically next word, all tape heads are reset to their initial positions, and the computation resumes in the initial state of the finite control[5] *.*

[5] A write-once tape may be of a single-head or double-head variety (see Section 2). In the latter case, the writing-head is not reset, but stays on the first blank tape cell.

We let D denote the class of sets decidable by write-once safe-storage machines where the total amount of space (work tape, safe storage and clock) is polynomially bounded.

Theorem 34 *Write-once safe-storage machines using polynomial space decide exactly Δ_2.*

Proof. Consider a polynomial-space computation by a safe-storage machine, and partition it at the points in time when the machine ticks the clock. In between ticks, the computation takes the contents of the clock z, the input x, and the contents of the safe-storage tape s and returns the new contents of the safe storage tape, say s'. This part of the computation can be simulated by a write-once Turing machine in polynomial space, so by Theorem 3 we find that the function $h(z, x, s) = s'$ is computable in polynomial time. Furthermore, since the safe-storage tape is write-once, h is a monotone function. Using the monotone recursion scheme we can easily conclude that every characteristic function of a language in D belongs to $\mathrm{r}\mathcal{A}_1 \simeq \Box_2$. Thus $D \subseteq \Delta_2$.

For the opposite inclusion, we simulate a polynomial number of queries to a Σ_1 oracle using the clock and the work tape. Let A be an arbitrary Σ_1 oracle, given by $A = \exists_p R$, where $R \in \mathrm{P}$, and let $\mathcal{M}$ be a Turing machine deciding some set in polynomial time with oracle A. We may assume without loss of generality that the maximal number m of oracle queries in any computation of $\mathcal{M}$ is a known polynomial of the size of the input, and similarly that all queries have a fixed size k that is also a known polynomial function of the input size. In order to simulate m queries $q_1, \ldots, q_m$ each of size k we use a $|m|+p(k)$-sized counter, divided into two parts i, y the first of size $|m|$ (sufficient for holding m in binary) and the second of size $p(k)$. We then simulate $\mathcal{M}$ using the safe storage tape, as in the proof of Theorem 2. When $\mathcal{M}$ makes the i-th query q_i, we tick the counter until it holds the word $i0^{p(k)}$. Then we keep ticking the counter, and for every tick we check whether $(y, q_i) \in R$, where y is taken from the counter and q_i from the safe-storage tape. This sub-computation is done using the write-once work tape, again as in Theorem 2. If we have a positive answer, then y witnesses that $q_i \in A$; we stop our search, write a 1-bit in the safe-storage tape as the answer from the oracle, and continue simulating $\mathcal{M}$. If the counter reaches the form $i1^{p(k)}$, and no witness was found, we know that $q_i \notin A$, and proceed with the simulation accordingly. We conclude that $\Delta_2 \subseteq D$. □

It may be interesting to note that we may remove the write-once work tape from the above machine model if we add an additional tape-head to the counter. We now sketch the technique for simulating the oracle in this case.

With two heads, and using only the finite control, it is possible to verify if the counter (or part of the counter) holds a valid computation history of a fixed non-deterministic machine $\mathcal{M}$ over input q_i (given on the safe-storage tape). This suffices for deciding if query q_i is in a Σ_1 set A: let $\mathcal{M}$ be a non-deterministic machine deciding A. Using the clock, we go through every

possible computation of $\mathcal{M}$ on input q_i, and find if any of these computations is accepting. Note that the total length of the computation history is bounded by a fixed polynomial and can, therefore, be precomputed on the safe-storage tape, in order to stop the search when a witness does not exist. Another way of doing that is to normalise $\mathcal{M}$ so that the last branch of its computation tree can be easily recognised.

Acknowledgements

Isabel Oitavem thanks FCT-UNL and CMAF-UL. Her research was supported by FEDER and FCT (project POCI/MAT/61720/2004 and Plurianual 2007). Bruno Loff thanks IST-UTL and CMAF-UL, where part of this work was carried out. The authors also thank the referee for the thoughtful reading and comments.

References

Stephen Bellantoni. Predicative recursion and the polytime hierarchy. In *FEASMATH: Feasible Mathematics: A Mathematical Sciences Institute Workshop*. Birkhauser, 1995.

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

Jin-Yi Cai and Merrick Furst. PSPACE survives constant-width bottlenecks. *International Journal of Foundations of Computer Science*, 2(1): 67–76, 1991.

Peter Clote. *Handbook of Computability Theory*, volume 140 of *Studies in Logic and the Foundations of Mathematics*, chapter 17 – Computation Models and Function Algebras, pages 589–681. Elsevier, 1999.

M. D. Gladstone. Simplifications of the recursion scheme. *Journal of Symbolic Logic*, 36(4):653–665, 1971.

Sandy Irani, Moni Naor, and Ronitt Rubinfeld. On the time and space complexity of computation using write-once memory or is pen really much worse than pencil? *Mathematical Systems Theory*, 25:141–159, 1992.

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.

Christos Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, Massachusetts, 1994.

Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976.