Characterizing Pspace with pointers

I. Oitavem[*]

Abstract

This paper gives an implicit characterization of the class of functions computable in polynomial space by deterministic Turing machines — *Pspace*. It gives an inductive characterization of *Pspace* with no ad-hoc initial functions and with only one recursion scheme. The main novelty of this characterization is the use of pointers (also called path information) to reach *Pspace*. The presence of the pointers in the recursion on notation scheme is the main difference between this characterization of *Pspace* and the well-known Bellantoni-Cook characterization of the polytime functions — *Ptime*.

1 Introduction

This paper gives an implicit characterization of *Pspace*, in the vein of the Bellantoni-Cook characterization of *Ptime* [3]. The Bellantoni-Cook characterization of *Ptime* [3] was established in numeric notation, however it can be rewritten over the word algebra $\mathbb{W}$. Here we proceed in an analogous way in order to describe an input-sorted term system, $\mathbf{STT}_{\mathbb{W}}$. In $\mathbf{STT}_{\mathbb{W}}$ we have only one recursion scheme which can be seen as the recursion on notation scheme of [3] plus pointers. The presence of pointers leads to a branching of the recursion, which generalizes the usual scheme of recursion on notation and allows the simulation of parallel polytime computations. We prove that $\mathbf{STT}_{\mathbb{W}}$ characterizes *Pspace*.

In contrast to the characterization of *Pspace* given in [8], here we use only one recursion scheme and no *ad-hoc* initial functions. The present work

[*]Dept. Matemática da FCT-UNL and CMAF-UL, e-mail: isarocha@ptmat.fc.ul.pt

also differs from the characterization of *Pspace* given by Leivant and Marion in [7]. We allow only a very restricted form of recursion substitution — the pointers. Moreover, instead of a (three) full sorted context, we work in a (two) input-sorted context.

This paper comes in the vein of previous work about *NC* (alternating polylog time and log space), [4]. There the concept of tree-recursion and, as a consequence of it, the introduction of pointers in the recursion scheme are used to characterize *NC*. The characterization given in [4] is established in a two-sorted context and, just like the one we give here, the pointers are in the highest tier, meaning that one can recurse on the pointers. Recent work showed that for *NC* we can avoid to recurse on the pointers, see [10].

2 The sorted term system $\mathbf{STT}_{\mathbb{W}}$

To define the term system $\mathbf{STT}_{\mathbb{W}}$ we consider the word constructors of $\mathbb{W}$, but we give a tree structure to the formulation of the recursion scheme. Therefore, **STT** stands for "Sorted Term system with Tree-recursion". In order to give a tree structure to the recursion scheme, we use pointers. Such a recursion scheme is substantially different from the standard recursion scheme defined over the word algebra $\mathbb{W}$, usually designated by recursion scheme on notation.

$\mathbb{W}$ is a word algebra generated by one nullary and two unary constructors, respectively, ϵ, $\mathbf{S}_0$ and $\mathbf{S}_1$. $\mathbb{W}$ can be interpreted over the set of all finite binary words. As usually in word algebra contexts, one considers a destructor (or predecessor) symbol of arity one — $\mathbf{P}$. One also introduces a symbol $\mathbf{C}$, of arity 4, for the conditional function of the algebra. They are defined as follows: $\mathbf{P}(\epsilon) = \epsilon$, $\mathbf{P}(\mathbf{S}_i x) = x$ and $\mathbf{C}(\epsilon, x, y, z) = x$, $\mathbf{C}(\mathbf{S}_i u, x, y_0, y_1) = y_i$, $i \in \{0, 1\}$.

The recursion scheme, that we consider, has the shape: $f(p, \epsilon, \bar{x}) = g(p, \epsilon, \bar{x})$ and $f(p, \mathbf{S}_i z, \bar{x}) = h(p, \mathbf{S}_i z, \bar{x}, f(\mathbf{S}_0 p, z, \bar{x}), f(\mathbf{S}_1 p, z, \bar{x}))$, $i \in \{0, 1\}$. The first input of f is called *pointer* or *path information.* A recursion scheme of the same sort is used in [4] to characterize *NC*. There, the starting algebra is $\mathbb{T}$ — the tree algebra generated by $\mathbf{0}$, $\mathbf{1}$ and $*$, of arity 0, 0 and 2 respectively. The designation of "path information" has there a literal meaning. We use it also here, nevertheless that now the path information is not a path.

We define a class of input-sorted function terms following notation introduced by Bellantoni and Cook in [3]. Function terms, in $\mathbf{STT}_{\mathbb{W}}$, have two sorts of input positions — *normal* and *safe*. As usual, we write normal and safe input positions by this order, and we separate them by a semicolon — $f(\bar{x};\bar{y})$.

Definition 1 $\mathbf{STT}_{\mathbb{W}}$ *is the smallest class of input-sorted functions which contains the constructors —* ϵ, $\mathbf{S}_0$, $\mathbf{S}_1$ *—, the destructor* $\mathbf{P}$*, conditional* $\mathbf{C}$ *and projection functions (over both input-sorts) and which is closed under the schemes of input-sorted composition and input-sorted tree-recursion: respectively,*

$$f(\bar{x};\bar{y}) = g(\bar{r}(\bar{x};);\bar{s}(\bar{x};\bar{y}))$$

and

$$\begin{aligned}
f(p,\epsilon,\bar{x};\bar{y}) &= g(p,\epsilon,\bar{x};\bar{y})\\
f(p,\mathbf{S}_0(z;),\bar{x};\bar{y}) &= h(p,\mathbf{S}_0(z;),\bar{x};\bar{y},f(\mathbf{S}_0(p;),z,\bar{x};\bar{y}),f(\mathbf{S}_1(p;),z,\bar{x};\bar{y}))\\
f(p,\mathbf{S}_1(z;),\bar{x};\bar{y}) &= h(p,\mathbf{S}_1(z;),\bar{x};\bar{y},f(\mathbf{S}_0(p;),z,\bar{x};\bar{y}),f(\mathbf{S}_1(p;),z,\bar{x};\bar{y})).
\end{aligned}$$

Of course, in the previous definition, it would be enough to introduce, as initial functions, the constructors, destructor and conditional having only safe argument positions. The correspondent functions over normal input positions could be then defined by input-sorted composition.

The main difference between the term systems $\mathbf{STT}_{\mathbb{W}}$ and $Ptime_{BC}$ — the Bellantoni-Cook characterization of *Ptime*, reformulated over the algebra $\mathbb{W}$ — is the recursion scheme. The recursion scheme of $\mathbf{STT}_{\mathbb{W}}$ generalizes the one in $Ptime_{BC}$ by doubling the critical argument of h and using pointers to distinguish them from each other. Therefore, one has that $Ptime_{BC}$ is trivially contained in $\mathbf{STT}_{\mathbb{W}}$. In particular, the concatenation and string product functions can be defined and iterated in $\mathbf{STT}_{\mathbb{W}}$:

1. We define a function $\oplus$ — concatenation — by input-sorted recursion (without pointers) such that[1] $|\oplus(y;x)| = |x| + |y|$.

$$\begin{aligned}
\oplus(\epsilon;x) &= x\\
\oplus(\mathbf{S}_0(y;);x) &= \mathbf{S}_0(;\oplus(y;x))\\
\oplus(\mathbf{S}_1(y;);x) &= \mathbf{S}_1(;\oplus(y;x))
\end{aligned}$$

[1]For x in $\mathbb{W}$, $|x|$ denotes the length of x, i.e. the number of $\mathbf{S}_0$ and $\mathbf{S}_1$ in x.

2. $\otimes$ — string product — is defined by input-sorted recursion (without pointers), with step function $\oplus$.

$$\otimes(\epsilon, x;) = \epsilon$$
$$\otimes(\mathbf{S}_0(y;), x;) = \oplus(x; \otimes(y, x;))$$

   One has $|\otimes(y, x;)| = |x| \cdot |y|$.

3. For any natural number $k > 1$, one may define by input-sorted composition:

$$\otimes^2(y_2, y_1;) = \otimes(y_2, y_1;)$$
$$\otimes^{k+1}(y_{k+1}, \cdots, y_1;) = \otimes^k(\otimes^2(y_{k+1}, y_k;), \cdots, y_1;)$$

   $\otimes^k$ is the string product function of arity k. Thus, $|\otimes^k(y_k, \cdots, y_1;)| = |y_1| \cdot \ldots \cdot |y_k|$.

Therefore, one has that:

Remark 1 *For any polynomial (with natural coefficients) q, there exists a term $t \in \mathbf{STT}_{\mathbb{W}}$ such that*[2] $\forall \bar{x}\ q(|\bar{x}|) = |t(\bar{x};)|$.

Notice that, in this framework, one can run a recursion over an output of a function which is itself defined by recursion, as done for $\otimes^k$. This will be important in the proof of lemma 3. That proof also uses the following:

Remark 2 *For any polytime function f there exists a function $\hat{f}$, in $\mathbf{STT}_{\mathbb{W}}$, and a monotone polynomial q_f such that $\forall \bar{w} \forall y \quad |y| \geq q_f(|\bar{w}|) \Rightarrow f(\bar{w}) = \hat{f}(y; \bar{w})$.*

This remark is a consequence of the recursion simulation lemma for $Ptime_{BC}$ — [2] and [3] — and the fact that $Ptime_{BC}$ is contained in $\mathbf{STT}_{\mathbb{W}}$, as class of input-sorted functions. The cited recursion simulation lemma states that for any polytime function f there exists a function $\hat{f}$, in $Ptime_{BC}$, and a monotone polynomial q_f such that $\forall \bar{w} \forall y \quad |y| \geq q_f(|\bar{w}|) \Rightarrow f(\bar{w}) = \hat{f}(y; \bar{w})$.

[2] $|\bar{x}| = (|x_1|, \cdots, |x_n|)$, where $\bar{x} = (x_1, \cdots, x_n)$.

3 $\mathbf{STT}_{\mathbb{W}}$ characterizes *Pspace*

We start by establishing a bounding lemma which is similar to the bounding lemma for *Ptime* given in [3]. However, here one has to take into account that, due to the pointers, during the recursion process the length of inputs in normal position may increase.

Lemma 1 *(Bounding lemma) If $f \in \mathbf{STT}_{\mathbb{W}}$ then there exists a polynomial q_f, with coefficients in $\mathbb{N}$, such that $\forall \bar{x}, \bar{y}\ |f(\bar{x};\bar{y})| \leq q_f(|\bar{x}|) + \max_i |y_i|$.*

Proof. The proof is by induction on the definition of f. For the initial functions the statement is trivial. If f is defined by input-sorted composition then one may consider $q_f(|\bar{x}|) = q_g(\bar{q}_{\bar{r}}(|\bar{x}|)) + \sum_i q_{s_i}(|\bar{x}|)$, where q_g, q_{r_j} and q_{s_i} are given by induction hypothesis. Finally, let f be given by input-sorted tree-recursion from g and h. Let q_g and q_h be given by induction hypothesis. Consider $q_f(|p|, |z|, |\bar{x}|) = |z| \cdot q_h(|p|+|z|, |p|+|z|, |\bar{x}|) + q_g(|p|+|z|, |p|+|z|, |\bar{x}|)$. It is straightforward to prove, by induction on the length of z, that for all p, z, $\bar{x}$ and $\bar{y}$ one has $|f(p, z, \bar{x}; \bar{y})| \leq q_f(|p|, |z|, |\bar{x}|) + \max_i |y_i|$. This finishes the proof. □

Lemma 2 $\mathbf{STT}_{\mathbb{W}}$ *is contained in Pspace.*

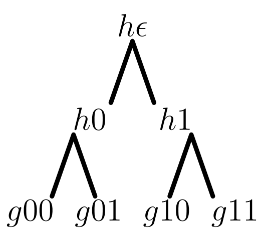
Proof. The proof is straightforward, using the previous lemma. For the recursion case, the machine description is analogous to the one given by Leivant and Marion, in the proof of lemma 4.7, in [7]. □

To prove the lower bound, i.e. that *Pspace* $\subseteq \mathbf{STT}_{\mathbb{W}}$, one uses the well-known characterization of *Pspace* via alternating Turing machines (ATMs):

Fact 1 *Let f be a function over $\mathbb{W}$. f is computable in polynomial space if, and only if, f is bitwise computable by an ATM in polynomial time, and $|f(w)|$ is polynomial in $|w|$.*

The concept of an ATM was introduced by Chandra, Kozen and Stockmeyer as a generalization of the non-deterministic Turing machine concept, see [5]. Here we use ATMs as described in [1], but we assume that they have only one tape. Thus, an ATM is a five-tuple $\langle Q, \Sigma, \delta, q_0, g \rangle$ where Q is the finite set of internal states, Σ is the tape alphabet, $\delta : Q \times \Sigma \to \mathcal{P}(\Sigma \times Q \times \{R, N, L\})$ is a partial function (the transition function), q_0 is the initial state and $g : Q \to \{\wedge, \vee, acc, rej\}$ is a function which partitions the states of Q into universal, existential, accepting and rejecting, respectively. We assume that non-terminating configurations have universal or existential states. To see if a ATM accepts an input x, we define a bottom-up labeling of its computation tree (or part of it) by the following rules: 1) the accepting leaves are labeled 1; 2) any existential node is labeled 1 if at least one of its sons has been labeled 1; 3) any universal node is labeled 1 if all its sons are labeled 1. The machine accepts the input if, and only if, the root is labeled 1.

Notice that if $f(\epsilon, 11;)$ is defined by recursion (with pointers) on its second input based on g and h, then one has the term $h(\epsilon; h(0; g(00;), g(01;)), h(1; g(10;), g(11;)))$ (some inputs are omitted), which corresponds to the tree



and it is suitable to carry out the bottom-up labeling described above (assuming that non-terminating configurations have two successor configurations). The first input of g and h is the pointer to the respective node, and it determines a path on the computation tree, i.e. a sequential computation. g should execute the computation determined by its first input and return 1 whenever it ends in an accepting configuration (otherwise returns 0). At this stage one has the tree above but with the leaves labeled 1 or 0 and we move one step up on the tree. h should execute the computation determined by its first input to see whether one reaches an existential or a universal state. Then, using the values returned by its last two inputs (i.e. the values returned by its left and right subtrees) h should return the label of the correspondent node according to the items 2) and 3) above. This process continues up to the root of the tree. What we described here for a tree of high 2 can be generalized and it will be used in the proof of the next lemma.

Maybe at this point one should also notice that, for instance, 01 determines a path in the tree above — branch first left (0), and then right (1) — leading to the leave labeled by $g(01;)$. If g is defined by recursion on 01 (i.e. on $\mathbf{S}_1(\mathbf{S}_0(\epsilon;);))$ then it can be used to iterate two different function terms (which can be associated with two transition functions of a given ATM), let us say g_0 (or g-left) and g_1 (or g-right). Then one gets $g(01;) = g_1(; g_0(\epsilon;))$ — apply first g-left, and then g-right. So, in both cases — following the path 01, or applying g-left/g-right — we do exactly the same thing: first left, and then right. In general, to use an input to run a recursion as above, or to determine a path on a tree, leads to the same left/right order.

Lemma 3 *ATMs working in polynomial time can be simulated in* $\mathbf{STT}_{\mathbb{W}}$.

Proof. Let M be an ATM which runs in time q, for some polynomial q in the length of the input. We are going to simulate M by $\mathbf{STT}_{\mathbb{W}}$ function terms. Let us assume that all internal states of the machine are codified by sequences of the same length. Codes of conjunctive, disjunctive, accepting and rejecting states end by 10, 00, 11 and 01, respectively, and they are placed at the end of the configurations. One may also assume that, for a given input x, all the configuration codes have the same length, $l(|x|)$, which is linear in $|x|$, and that all non-terminating configurations have two successor configurations. Therefore, we can split the transition function of M, δ, in δ_0 and δ_1. Let $c(x;)$ be the code of the initial configuration and let $t(x;)$ be a $\mathbf{STT}_{\mathbb{W}}$ term such that $|t(x;)| = q(|x|)$. This is possible since q is a polynomial, cf. remark 1. For $i \in \{0, 1\}$, one may consider polytime computable functions Δ_i, which for a given configuration code w return the next configuration code according to δ_i — notice that, in order to transform a configuration code in its next configuration code, one just has to read the given configuration code and to change in it a fixed number of bits (according to δ_i). Thus, by remark 2, there exists a function $\hat{\Delta}_i$ in $\mathbf{STT}_{\mathbb{W}}$ and a polynomial q_{Δ_i} such that $\forall w\ \forall y\ \ |y| \geq q_{\Delta_i}(|w|) \Rightarrow \Delta_i(w) = \hat{\Delta}_i(y; w)$. Replacing, in the previous expression, y by $L_{\Delta_i}(x;)$ one has that $\forall w\ \forall x\ \ |L_{\Delta_i}(x;)| \geq q_{\Delta_i}(|w|) \Rightarrow \Delta_i(w) = \hat{\Delta}_i(L_{\Delta_i}(x;); w)$, where L_{Δ_i} is a $\mathbf{STT}_{\mathbb{W}}$ term as follows. Given an input x, all configuration codes w verify $|w| = l(|x|)$ where l is linear in $|x|$. Thus, $q_{\Delta_i}(|w|)$ is equal to $(q_{\Delta_i} \circ l)(|x|)$. The composition of a polynomial with a linear function is a polynomial, and so $q_{\Delta_i} \circ l$ is a polynomial in $|x|$. Therefore, by remark 1, there exists a function term L_{Δ_i} in $\mathbf{STT}_{\mathbb{W}}$ such that $|L_{\Delta_i}(x;)| = (q_{\Delta_i} \circ l)(|x|)$. Now, recalling that $q_{\Delta_i}(|w|)$ is $(q_{\Delta_i} \circ l)(|x|)$, one has $|L_{\Delta_i}(x;)| = q_{\Delta_i}(|w|)$ (thus, a fortiori $|L_{\Delta_i}(x;)| \geq q_{\Delta_i}(|w|)$). Therefore, for any input x, given a configuration code w, $\Delta_i(w) = \hat{\Delta}_i(L_{\Delta_i}(x;); w)$ where $\hat{\Delta}_i$ and L_{Δ_i} are in $\mathbf{STT}_{\mathbb{W}}$. This means that (reusing the symbol Δ_i) we can consider a function term $\Delta_i(x; w) = \hat{\Delta}_i(L_{\Delta_i}(x;); w)$ in $\mathbf{STT}_{\mathbb{W}}$, which for any input x and a given configuration code w returns the next configuration code according to δ_i.

Let us define some auxiliary function terms: *RUN* and *RETURN*.

RUN is defined by recursion (without path information). For a path p and a (initial) configuration code $c(x;)$, *RUN* simulates the (sequential) computation performed by M along the branch p starting with the configuration code $c(x;)$.

$$\begin{aligned} RUN(\epsilon, x;) &= c(x;) \\ RUN(\mathbf{S}_0 p, x;) &= \Delta_0(x; RUN(p, x;)) \\ RUN(\mathbf{S}_1 p, x;) &= \Delta_1(x; RUN(p, x;)) \end{aligned}$$

RETURN is defined by composition. It gives the value returned by a configuration of code u assuming that its successor configurations, if there are any, return i and j.

$$RETURN(; i, j, u) = C(; u, \epsilon, C(; P(; u), \epsilon, i \vee j, i \wedge j), C(; P(; u), \epsilon, 0, 1)),$$

where $i \vee j = C(; i, \epsilon, C(; j, \epsilon, 0, 1), 1)$ and $i \wedge j = C(; i, \epsilon, 0, C(; j, \epsilon, 0, 1))$. One has that: $i \vee j$ is 0 if i and j are 0, and is 1 otherwise; $i \wedge j$ is 1 if i and j are 1, and is 0 otherwise. Of course, by 0 and 1 we mean $\mathbf{S}_0(\epsilon;)$ (or $\mathbf{S}_0(;\epsilon)$) and $\mathbf{S}_1(\epsilon;)$ (or $\mathbf{S}_1(;\epsilon)$), respectively.

Let us consider the function f defined by input-sorted recursion over $\mathbf{STT}_{\mathbb{W}}$:

$$\begin{aligned} f(p, \epsilon, x;) &= RETURN(; \epsilon, \epsilon, RUN(p, x;)) \\ f(p, \mathbf{S}_0 z, x;) &= RETURN(; f(\mathbf{S}_0 p, z, x;), f(\mathbf{S}_1 p, z, x;), RUN(p, x;)) \\ f(p, \mathbf{S}_1 z, x;) &= RETURN(; f(\mathbf{S}_0 p, z, x;), f(\mathbf{S}_1 p, z, x;), RUN(p, x;)). \end{aligned}$$

One has that $M(x) = f(\epsilon, t(x;), x;)$. $\square$

Lemma 4 *Pspace is contained in* $\mathbf{STT}_{\mathbb{W}}$.

Proof. Let F be a *Pspace* function. By fact 1 F is bitwise computable by an ATM, M, running in polynomial time. By the previous lemma one knows that M can be simulated by function terms in $\mathbf{STT}_{\mathbb{W}}$. Thus, one ensures that there exists $f \in \mathbf{STT}_{\mathbb{W}}$ such that $f(i, \bar{x};) = |i|$'th bit of $F(\bar{x})$. Now, noticing that the outputs of *Pspace* functions are polynomially bounded and attending to remark 1, the definition of the function $F \in \mathbf{STT}_{\mathbb{W}}$ by recursion (without pointers) is straightforward. $\square$

From lemma 2 and lemma 4 one concludes that

Theorem 1 $\mathbf{STT}_{\mathbb{W}}$ *characterizes Pspace.*

Therefore, one has an implicit characterization of *Pspace* that is closely related to the well-known Bellantoni-Cook characterization of *Ptime* — *Ptime*$_{BC}$ [3]. $\mathbf{STT}_{\mathbb{W}}$ results from *Ptime*$_{BC}$ by giving a tree structure to the recursion on notation scheme. This is achieved by allowing, in the usual recursion on notation scheme, a specific form of substitution. The substitution parameter is the pointer (or path information) and it leads to a branching of the recursion which allows us to simulate parallel polytime computations.

4 An unsorted variant

A similar characterization of *Pspace* can be achieved working in a non-sorted context, by use of explicit bounds on the recursion scheme. Such a characterization, here denoted by $\mathbf{BTT}_{\mathbb{W}}$, is the parallel of Cobham's characterization of *Ptime* [6], now for *Pspace*.

Definition 2 $\mathbf{BTT}_{\mathbb{W}}$ *is the smallest class of functions which contains the constructors —* $\epsilon, \mathbf{S}_0, \mathbf{S}_1$ *—, the destructor* $\mathbf{P}$*, conditional* $\mathbf{C}$ *and projection functions and which is closed under the schemes of composition and bounded tree-recursion: respectively,*

$$f(\bar{x}) = g(\bar{h}(\bar{x}))$$

and

$$\begin{aligned} f(p,\epsilon,\bar{x}) &= g(p,\epsilon,\bar{x}) \\ f(p,\mathbf{S}_0(y),\bar{x}) &= h(p,\mathbf{S}_0(y),\bar{x},f(\mathbf{S}_0(p),y,\bar{x}),f(\mathbf{S}_1(p),y,\bar{x}))_{|t(p,\mathbf{S}_0(y),\bar{x})} \\ f(p,\mathbf{S}_1(y),\bar{x}) &= h(p,\mathbf{S}_1(y),\bar{x},f(\mathbf{S}_0(p),y,\bar{x}),f(\mathbf{S}_1(p),y,\bar{x}))_{|t(p,\mathbf{S}_1(y),\bar{x})}, \end{aligned}$$

where t *is a function explicitly definable from* ϵ*,* $\mathbf{S}_0$*,* $\mathbf{S}_1$*, string concatenation and string product.*

$x_{|y}$ denotes x truncated to the length of y. From the definition of the functions bounding the recursion, it is obvious that they are polynomial in the length of the inputs. Therefore, one trivially has that

Lemma 5 *For all* $f \in \mathbf{BTT}_{\mathbb{W}}$ *there exists a polynomial* q_f*, with coefficients in* $\mathbb{N}$*, such that* $\forall \bar{x}\ |f(\bar{x})| \leq q_f(|\bar{x}|)$*.*

This lemma is essential to ensure the upper bound.

Lemma 6 $\mathbf{BTT}_{\mathbb{W}}$ *is contained in Pspace.*

Proof. The proof is similar to the proof of lemma 2, using the lemma 5 above. $\square$

The lower bound is almost immediate.

Lemma 7 *Pspace is contained in* $\mathbf{BTT}_{\mathbb{W}}$*.*

Proof. By lemma 4 we have that *Pspace* is contained in $\mathbf{STT}_{\mathbb{W}}$. Therefore, to prove that *Pspace* is contained in $\mathbf{BTT}_{\mathbb{W}}$, we just have to show that $\mathbf{STT}_{\mathbb{W}}$ is contained in $\mathbf{BTT}_{\mathbb{W}}$. That means: for all $F \in \mathbf{STT}_{\mathbb{W}}$ there exists $f \in \mathbf{BTT}_{\mathbb{W}}$ such that $\forall \bar{x}, \bar{y}\ F(\bar{x}; \bar{y}) = f(\bar{x}, \bar{y})$. The proof is straightforward by induction on the definition of F. The only non trivial case occurs when F is defined by input-sorted tree-recursion, from G and H. In this case we define f by bounded tree-recursion from g, h and t. g and h are functions given by induction hypothesis. t is any function explicitly definable from ϵ, $\mathbf{S}_0$, $\mathbf{S}_1$, string concatenation and string multiplication such that $p_F(|\bar{x}|, |\bar{y}|) \leq |t(\bar{x}, \bar{y})|$, for some polynomial p_F verifying $\forall \bar{x}, \bar{y}\ |F(\bar{x}; \bar{y})| \leq p_F(|\bar{x}|, |\bar{y}|)$. The existence of such a polynomial is a consequence of the bounding lemma for $\mathbf{STT}_{\mathbb{W}}$ functions — lemma 1. $\square$

Thus, we have that

Theorem 2 $\mathbf{BTT}_{\mathbb{W}}$ *characterizes Pspace.*

This characterization of *Pspace* is machine independent, but not resource independent. The bound of the recursion scheme is an explicit reference to the polynomial resource constraints. However, it shows that the sorted approach to *Pspace* developed in this paper can be reformulated in different contexts. In this last section we reformulated it in a bounded context only, but it is clear that it also holds in other contexts. Namely, an unsorted version of the term system introduced here can be used, together with an appropriated measure of complexity (rank), to characterize *Pspace*. By an appropriated rank we mean, for instance, the rank $\hat{\sigma}$ described in [9].

The unsorted version presented here might be of interest for further research, namely for discussing induction schemes in the context of bounded arithmetic.

Acknowledgements

Research done while staying at the Mathematisches Institut der Universität München. I like to thank LMU, my home university (UNL), CMAF and FCT (grant SFRH/BPD/9455/2002 and project POCI/MAT/61720/2004).

References

[1] Balcázar J. L., Díaz J., Gabarró J.: Structural Complexity II. Springer-Verlag (1990)

[2] Bellantoni S.: Predicative Recursion and Computational Complexity. Ph. D. Dissertation, University of Toronto (1993)

[3] Bellantoni S., Cook S.: A New Recursion-Theoretic Characterization of Polytime Functions. Computational Complexity **2**, pp.97-110 (1992)

[4] Bellantoni S., Oitavem I.: Separating NC along the δ axis. ICC Special issue of TCS **318**, pp.57-78 (2004)

[5] Chandra A. K., Kozen D. C., Stockmeyer L. J.: Alternation. J.ACM, pp.114-133 (1981)

[6] Cobham A.: The intrinsic computational difficulty of functions. Proc. of the 1964 International Congress for Logic, Methodology, and the Philosophy of Science, ed. Y. Bar-Hillel, North Holland, Amsterdam, pp.24-30 (1965)

[7] Leivant D., Marion J.: Ramified Recurrence and Computational Complexity II: Substitution and poly-space. LNCS 993, 8th Proceedings of CSL, pp.486-500 (1994)

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

[9] Oitavem I.: Implicit characterizations of Pspace. Proof Theory in Computer Science, LNCS 2183, ed. R. Kahle et al., Springer, pp.170-190 (2001)

[10] Oitavem I.: Characterizing NC with tier 0 pointers. Mathematical Logic Quarterly **50**, N.1, pp.9-17 (2004)

[11] Thompson D.: Subrecursiveness: Machine-Independent Notions of Computability in Restricted Time and Storage. Mathematical Systems Theory **6**, N.1, pp.3-15 (1971)