ELSEVIER

Theoretical Computer Science 250 (2001) 83–99

Theoretical Computer Science

www.elsevier.com/locate/tcs

A predicative and decidable characterization of the polynomial classes of languages ☆

S. Caporaso [a,*], M. Zito [b], N. Galesi [c]

[a] *Univ. di Bari, Dip. di Informatica, v. Amendola 173, I-70126, Italy*
[b] *University of Warwick, Department of Computer Science, Coventry, CV4 7AL, UK*
[c] *Univ. Politecnica de Catalunya, Dept. LSI, Barcelona, Spain*

Received September 1995; revised October 1997
Communicated by J. Díaz

Abstract

Characterizations of PTIME, PSPACE, the polynomial hierarchy and its elements are given, which are *decidable* (membership can be decided by syntactic inspection to the constructions), *predicative* (according to points of view by Leivant and others), and are obtained by means of increasing restrictions to course-of-values recursion on trees (represented in a dialect of Lisp).

Keywords: Computational complexity; Predicative recursion; Functional programming; Lisp

1. Introduction

1.1. Impredicativity

The *naive comprehension principle* states that a set exists for every collection of sets admitting a description. According to Poincaré [14, p. 307], a description of entity E is *impredicative* if it uses a variable whose domain includes E. Impredicativity and comprehension, when united, produce sets of increasing size (e.g., $Pow(x) := \{y \mid y \subseteq x\}$), and may lead to paradoxes. In ramified set theory comprehension is ruled by means of *stages* [16]. All sets used to form a new set must have been formed at earlier stages. If $\mathcal{N}$ has been formed at α, then the real numbers $\mathcal{R}$ exist only at stages $\beta > \alpha$. Moving by levels from $\mathcal{N}$ to $\mathcal{R}$ is the kernel of *predicative* [9, 17] *analysis*. In particular, we

☆ The second and third co-authors mainly contributed to this paper when they were post-graduate students at Bari, with the financial support of Dip. d' Informatica and of the Italian MURST "funds 60%".
* Corresponding author.
E-mail addresses: caporaso@di.uniba.it (S. Caporaso), m.zito@dcs.warwick.ac.uk (M. Zito), galesi@lsi.up.es (N. Galesi).

0304-3975/01/$ - see front matter
PII: S0304-3975(99)00116-4

used [6] infinitary proof-theoretical methods and a relativized programming language to study the sets reachable within ω_1^{ck} — the Church–Kleene limit for the constructive ordinals. Since 1990, complexity is indebted to Leivant for discovering the analogy between, on one side, growth of sets, predicativity and comprehension, and, on the other, growth of functions and nested recursion.

A primitive recursive (PR) description of class (function) f like

$$f := \{(x+1, y) \mid \exists z[(x,z) \in f \ \textit{and} \ (x,z,y) \in h]\}$$

is impredicative, though harmless in itself. The critical point is reached when comprehension is adopted to take that description into a *definition*, thus implying the perfect existence of f, and, via further PR's, of very large functions.

1.2. Recursion schemes

Function $f(\mathbf{x}, y)$ is defined by PR on *notations* (PRN) in (the step-function) $h(\mathbf{x}, y, s)$, with parameters $\mathbf{x}$ and principal (auxiliary) variable $y(s)$ if we have $f(\mathbf{x}, y) = h(\mathbf{x}, y, f(\mathbf{x}, \lfloor y/2 \rfloor))$. f is defined by PRN with *parameter substitution* (PRNPS) in h and in the tuple $\mathbf{g}$ if we have $f(\mathbf{x}, y) = h(\mathbf{x}, y, f(\mathbf{g}(\mathbf{x}, y), \lfloor y/2 \rfloor))$. Finally, f is defined by *full course-of-values* recursion (VR) in h and in the *decreasing functions* $d_i, \ldots, d_q$ if we have $d_i(y) < y$ and $f(\mathbf{x}, y) = h(\mathbf{x}, f(\mathbf{x}, d_1(y)), \ldots, f(\mathbf{x}, d_q(y)))$. We say that a function is (essentially) *boolean* if its range consists of objects of length 1.

1.3. Main result

Boolean PR and full VR are equivalent, *with respect to* the class of all elementary functions, in the sense that this class is closed under both schemes. Since in [7], we show that PTIMEF is not, and all sub-elementary classes do not appear to be closed under full VR, we introduce in this paper a sequence VR_3, VR_2, VR_1 of progressive retrictions to full VR. Let $\mathrm{VR}_i(\mathscr{C})$ denote the class of all functions which can be defined: (a) by a single application of VR_i to functions in $\mathscr{C}$; or (b) by substitution inside functions in $\mathrm{VR}_i(\mathscr{C})$ of functions from a class $\mathscr{I}$ of initial functions over a tree algebra $\mathscr{A}$ (which includes an analogue, called *clone*, of $\mathit{smash}(x, y) := 2^{|x| \cdot |y|}$); we have

$$\mathrm{PTIME} = \mathrm{VR}_1(\mathscr{I}); \ \ \Sigma_n = \mathrm{VR}_2(\Delta_n); \ \ \Delta_{n+1} = \mathrm{VR}_1(\Sigma_n); \ \ \mathrm{PSPACE} = \mathrm{VR}_3(\mathrm{PTIME}).$$

Since only one recursion is needed to define PTIMEF and to move to next class, these characterizations may be viewed as analogues for the polynomial classes of Kleene's normal form for PR functions. However, to give to our characterizations a chance as *programming languages*, in the sense discussed below, we show that PTIME is closed under VR_1, and PSPACE under VR_1, VR_3.

In schemes VR_i no distinction between variables is made; by certain purely syntactic conditions, such schemes produce boolean functions only. The general idea is to use *clone* in order to assign the variables with initial values large enough; at each recursive step, a single digit is returned, but, at the same time, some constructors are applied

to certain variables and some destructors to certain others (not necessarily the same at each step). The key condition is that the overall balance of such con/destructions must be negative. In the Conclusion we discuss *rationale* under the choice of these schemes (in the following referred to as SVR,OR-VR,FVR) and their naturalness.

1.4. Contrast with literature

In [12] Leivant translates his original proof theoretic formulation [11] in terms of a programming language; the idea was to assign a lower *tier* to the auxiliary variable of a PRN than to the principal. If for example, s' is used as step function to define addition by $+(x', y) := +(x, y)'$, then $tier(x) > tier(y)$. This avoids using $+(s, s)$ as step function in $2^{y+1} := 2^y + 2^y$; the reason is that $+(s, s)$ is defined by substitution of a same variable for two differently tiered arguments. The independent approach by Bellantoni and Cook [1, 3] was more different in principle than in practice. A distinction is defined between *safe* and *unsafe variable*: all recursion variables of previous PRN's are unsafe, while the auxiliary variable of every recursion must be safe. PTIMEF is defined by closure under *safe* PRN and under a restricted form of substitution.

Recently [4], PTIMEF with the Grzegorczyck hierarchy $\mathscr{E}^n$ at and above the elementary level $\mathscr{E}^3$, have been harmonized by counting the number of times an unsafe variable is used as principal. The same harmonization was in [5] (together with a characterization of $\mathscr{NP}$) by means of a new machine model of predicative computation. According to a comment by Bellantoni, "the nesting depth of iterations in this model corresponds to the number of tiers in a multi-tiered recursion system such as that of Leivant." A main difference was that Leivant's tiers collapse at level 2, while in this system depth $n = 1, 2, 3, \ldots$ characterizes LINTIMEF, PTIMEF, $\mathscr{E}^3, \ldots$.

The first predicative characterization of a space class is the definition of LINSPACEF in Bellantoni's thesis; he showed that when "unary numerals [are] used, the functions definable by ramified recurrence are precisely the ones computable in linear space" [12]. In [13], Leivant and Marion characterize PSPACEF by using tiered PRNPS. In [2] Bellantoni obtains the polynomial hierarchy PH by adding *un-bounded* minimization to PTIMEF — a striking confutation of the widely accepted asymmetry: arithmetic hierarchy = (PR, $\neg$, unbounded $\exists$) versus PH = (PTIME, $\neg$, bounded $\exists$). We are not aware of predicative characterizations of the classes of languages PTIME, PSPACE. A difficulty with the classes of languages is that they are not inductive: if f is an acceptor, defined by PR in g and h, function h does not need to be an acceptor too.

The proof that scheme VR_i characterizes a class $\mathscr{C}_i$, which was already characterized by the recursion scheme R_i, implies the equivalence of R_i and VR_i with respect to $\mathscr{C}_i$. Besides its possible interest on its own, this kind of results may be regarded as adding robustness to the predicative approach.

Instead of mere *extensional equality* between functions, let us consider the *intension* implicit in their algorithms. In this case, each VR_i may be regarded as *intensionally stronger with respect to* $\mathscr{C}_i$ than the homologue schemes R_i defining predicatively class $\mathscr{C}_i$ in the reported literature. Indeed each R_i is a restriction of VR_i, and this statement

holds even if we ignore: (a) the difference between trees and words; and (b) the fact that all our variables have the same status, since all of them may be recursed upon in a same recursion. For example: in safe or tiered PRN $f(x)$ depends on the single value $f(\lfloor x/2 \rfloor)$, instead than on an n-ple of the form $f(\lfloor x/2^{t_1} \rfloor), \ldots, f(\lfloor x/2^{t_q} \rfloor)$ $(t_1, \leqslant t_q \geqslant 1)$. In PRNPS we may change the parameters, not the previous value of the principal variable. A separation problem $\mathscr{C}_1 \subset ?\mathscr{C}_2$ might be better understood if we compare the strongest scheme for $\mathscr{C}_1$ with the weakest one for $\mathscr{C}_2$.

The program of revisiting complexity in a predicative framework appears to be now involving several scholars. An approach to complexity in terms of predicative operators should aim at collecting as many classes as possible in a *unified taxonomy*, based on the same and single criterion. After all, the approach in terms of machines and resources *actually is* a taxonomy. In [1, 2, 4] several classes are indeed reduced to the same rule; but LINSPACEF does not match it, and PSPACEF is missing. In [12, 13] we can compare two important classes, but PH is missing in this case.

Insight to a class $\mathscr{C}$ could be increased by a programming language allowing an easy representation of the single elements of $\mathscr{C}$. Notations, as well as λ-calculus, are not the best way, for both authors and readers, to show the structure of an algorithm. For example (cf. Note 22), we did not succeed in extending our characterization of PSPACE to a word-algebra (without nested recursion).

$\mathscr{A}$ is introduced and handled in the usual mathematical mode, but notations are curbed in a way allowing anybody familiar with Lisp to immediately translate all schemes and functions into the equivalent constructs and programs of this language. Though no previous knowledge of Lisp is needed to read this paper, we feel that it may contribute in gaining to complexity a language which offers the obvious advantages of its high level, and suits, at the same time, mathematical methods of investigation like induction on constructions and on data.

2. Preliminaries

A *list* is an element of the tree algebra without empty word which is defined by $r \geqslant 2$ 0-*ary generators* $T, F, \ldots$ (called *atoms*, and denoted by sequences of capital letters and digits) and by the *binary generator* $\circ$.

We write (x, y) for $\circ xy$ and $(x_1, \ldots, x_n)$ for $\circ x_1(x_2, \ldots, x_n)$ $(n \geqslant 3)$. x_i is the ith component of list $y = (x_1, \ldots, x_n)$, and n is its *number of components*. A list is *simple* if all its components are atoms. Special roles are assigned to atoms T and F (the truth-values *true* and *false*) and to the simple list $nil := (F, F)$ (see Example 3). For example, *nil* has two components, and (nil) is not a list.

In Note 25 we show that two atoms are enough for all results in this paper.

Notation 1. Without further notice, except for emphasis:
1. $i, j, m, \ldots, r$ are natural numbers; a, b are atoms; $s, \ldots, z, s_1, \ldots$ are lists.
2. $f, g, h, f_1, \ldots$ are functions, taking lists into lists; in particular, $d, d_1, \ldots$ are unary.

By a composite notation like $f[x_1;\ldots;x_n]$ we mean that the actual arguments of f occur among the x's.

3. Whenever a variable E has been defined on a class C of syntactical entities, $\mathbf{E}, \mathbf{E}^1, \ldots$ are tuples of elements in C; if $\mathbf{E}^j$ has been already introduced in the discourse, then E_i^j is its ith element, if any.

For example **a**, **x**, **f**, **d** are tuples of atoms, lists, functions and unary functions. If $\mathbf{a}^4$ is A; B; A then a_2^4 is B, and a_4^4 does not exist.

Definition 2. (1) The *length* $|x|$ of x is the number of atoms occurring in (the value assigned to) x. $|f[\mathbf{x}]|$ is the length of $f[\mathbf{x}]$ when a system of values is assigned to $\mathbf{x}$. The length $|\mathbf{E}|$ of the tuple of variables or functions $\mathbf{E}$ is $\sum_i |E_i|$.

(2) Tuple $\mathbf{y}$ is *minimal* if some y_i is an atom.

(3) For every n, define the following partial orders on the n-ples

$\mathbf{u} \prec_\mu \mathbf{w}$ iff $\mathbf{u}$ is minimal, or both $\mathbf{u}$ and $\mathbf{w}$ are not minimal and $|\mathbf{u}| < |\mathbf{w}|$;

$\mathbf{u} \prec_{occ} \mathbf{w}$ iff for each $1 \leqslant i \leqslant n$ we have that u_i occurs in w_i.

We show substitutions in the most rudimental way, by just replacing the substituted variables with the substituted functions. In this way we get simpler definitions, at the price of a systematic ambiguity between functions and values: deciding whether $f[x]$ and $g[y]$ are *the same thing* is left to context.

Notation 3. Given an n-ple $\mathbf{d}$ of unary functions, and an n-ple $\mathbf{y}$, we write $\mathbf{d}[\mathbf{y}]$ for the n-ple $d_1[y_1];\ldots;d_n[y_n]$.

Thus, given the n-ples $\mathbf{d}[u]$ and $\mathbf{y}$ together with function $f[\mathbf{y}]$, we have

$$f[\mathbf{d}[\mathbf{y}]] = f[d_1[y_1];\ldots;d_n[y_n]].$$

2.1. Initial functions

The class $\mathscr{I}$ of all *initial* functions is the closure under substitution of:

(1) the *predicates* $x = y$ and $at[x]$, which are *true* iff x and y are the same atom, and, respectively, x is an atom;

(2) the *constructors app* (*appending* its second argument to the first), *cons*, *clone*; and the *destructors* H and T, returning the *head* and *tail* of their argument:

$$app[(x_1,\ldots,x_m);(x_{m+1},\ldots,x_{m+n})] = (x_1,\ldots,x_{m+n}); \quad cons[x;y] = (x,y);$$

$$clone[x;(y_1,\ldots,y_n)] = (x,\ldots,x)\ (n \text{ times});$$

$$\mathrm{H} \circ xy = x; \quad \mathrm{T} \circ xy = y; \qquad \mathrm{H}[a] = \mathrm{T}[a] = a;$$

we often write Hx for H$[x]$, and H$\mathbf{x}$ for H$x_1;\ldots;$Hx_n; similarly for T;

(4) the *conditional* $cond[x;y;z] = y(z)$ iff $x = (\neq) T$;

(5) the *identity* $id[x]=x$, and, for all list y, the *unary constant function* $y[x]$; we often let these functions be replaced by their results (for example, we write $not[x]:=cond[x;F;T]$ (instead of $cond[id[x];T[x];F[x]]$).

2.2. Decreasing tuples

Notation 4. (1) $(x)_i^n$ is short for $\mathrm{H}\mathrm{T}^{i-1}x$ $(1\leqslant i<n)$ (that is H applied to the result of $i-1$ T's), and for $\mathrm{T}^{n-1}[x]$ $(n=i)$; for $1\leqslant i\leqslant n$ we have $(x_1,\ldots,x_n)_i^n=x_i$; we omit the superscript when n is known.

(2) For every atom a, the *prefix* $\langle a;y\rangle$ is the unary function returning $app[a;y]$ if y is not an atom, and y itself otherwise.

Definition 5. A *decreasing tuple* is an n-ple $\mathbf{d}$ of unary functions, such that

1. each d_i $(i\leqslant n)$ is in the form $d_{i1}[d_{i2}[id[x]]$, where d_{i1} is a sequence of $n_i\geqslant 0$ prefixes, d_{i2} is a sequence of $m_i\geqslant 0$ destructors, and id is present only if both d_{ij} are absent.
2. its *rate of growth* $\gamma(\mathbf{d}):=\sum_{1\leqslant i\leqslant n}(n_i-m_i)$ is negative.

Let $\mathbf{d}$ be decreasing. We have that either $\mathbf{y}$ is minimal, and then so it is $\mathbf{d}[\mathbf{y}]$ too, or that $|\mathbf{d}[\mathbf{y}]|=|\mathbf{y}|+\gamma(\mathbf{d})$. This proves the

Lemma 6. *If* $\mathbf{d}$ *is a decreasing tuple and* $\mathbf{y}$ *is not minimal, we have* $\mathbf{d}[\mathbf{y}]\leqslant_\mu\mathbf{y}$.

2.3. Boolean functions

Definition 7. The class $\mathscr{B}$ of all *boolean functions* consists of all $f[\mathbf{x}]$ such that each occurrence of every x_i is in the scope of a predicate or of a constant function of the form $a[x_i]$.

Note 8. $f\in\mathscr{B}$ obviously implies that $f[\mathbf{y}]$ is an atom. Though such atoms are not necessarily truth-values, as the term "boolean" would suggest, we see from Note 25 that this point may be regarded as inessential.

Definition 9. (1) Function $f[\mathbf{x}]$ is defined by *substitution of* $g[\mathbf{x}]$ *inside* $h[\mathbf{x};y]$ if we have $f[\mathbf{x}]=h[\mathbf{x};g[\mathbf{x}]]$.

(2) For every class $\mathscr{C}$ of functions, $\mathscr{C}^{\#}$ is the class of all functions defined by substitution of functions in $\mathscr{I}$ inside a function $\in\mathscr{C}\cup\mathscr{C}^{\#}$.

Notice that, for every $\mathscr{B}_0\subseteq\mathscr{B}$ we have $\mathscr{B}_0^{\#}\subseteq\mathscr{B}$. In other words, substitutions of initial functions inside boolean functions yield boolean functions.

Example 1. In $\mathscr{B}$ we may define x or $y:=cond[x;T;cond[y;T;F]]$ and x and $y:=cond[x;\ cond[y;T;F];F]$.

3. Course-of-values recursion

We now introduce three VR schemes. The main difference is that in the first the value $f[\mathbf{y}]$ of the function being defined may depend on q values $f[\mathbf{d}^1[\mathbf{y}]],\ldots,f[\mathbf{d}^q[\mathbf{y}]]$ such that $\mathbf{d}^i[\mathbf{y}] \prec_\mu \mathbf{y}$, for all i; hence the overall number of repetitions of the step function may be exponential in $|\mathbf{y}|$. In the second, the form of the step function is drastically restricted to an obvious equivalent of non-determinism. In the last one, we may choose between: $q=1$; or else $q\geqslant 1$ and tuples $\mathbf{d}^i$ such that for all $i\leqslant q$ we have $\mathbf{d}^i[\mathbf{y}] <_{occ} \mathbf{y}$ (instead of $\prec_\mu$).

Definition 10. (1) Assume given
(a) r *principal variables* $\mathbf{y}$, and q *auxiliary variables* $\mathbf{s}$;
(b) a *basis function* $g[\mathbf{y}]$ and a *step function* $h[\mathbf{y};\mathbf{s}]$, both boolean;
(c) q decreasing r-ples $\mathbf{d}^1,\ldots,\mathbf{d}^q$.
(2) Function f is defined by
(a) *short course-of-values recursion* (SVR) *in* g,h, with *decreasing tuples* $\mathbf{d}^i$ if we have

$$f[\mathbf{y}] = \begin{cases} g[\mathbf{y}] & \text{if } \mathbf{y} \text{ is minimal,} \\ h[\mathbf{y}; f[\mathbf{d}^1[\mathbf{y}]];\ldots; f[\mathbf{d}^q[\mathbf{y}]]] & \text{otherwise.} \end{cases}$$

(b) *or-course-of-values recursion* if it is defined by SVR in the form above, and the form of the step function is s_1 *or* ... *or* s_q;
(c) *fast course-of-values recursion* (FVR) if it is defined by SVR in the form above, and either $q=1$, or $q>1$ and no prefix occurs in any $\mathbf{d}^i$.

Example 2. Define the equality between lists by

$$equal[y_1; y_2] = \begin{cases} y_1 = y_2 & \text{if } \mathbf{y} \text{ is minimal,} \\ equal[\mathrm{H}\mathbf{y}] \text{ and } equal[\mathrm{T}\mathbf{y}] & \text{otherwise.} \end{cases}$$

equal is FVR in the basis function $y_1=y_2$, and in the step function $h[\mathbf{y};\mathbf{s}]=s_1$ and s_2, with the two decreasing couples $d_j^1[u]=\mathrm{H}u$ and $d_j^2=\mathrm{T}u$ $(j=1,2)$, which consist of destructors.

In the following we use the SVR and OR-VR schemes to characterize PSPACE and NP. However, the essence of the related proofs may be seized by the following examples showing how the acceptors for a PSPACE- and for an NP-complete language may be defined by SVR or OR-VR, applied to basis functions in PTIME.

Notation 11. To improve readability we often display a definition like

$$h[x] := cond[f_1[x]; g_1[x]; cond[\ldots; cond[f_n[x]; g_n[x]; g_{n+1}[x]]\ldots]]$$

by means of a, so to say, *definition by cases* of the form

$$h[x] := \begin{cases} g_1[x] & \text{if } f_1[x] \\ \dots & \dots \\ g_n[x] & \text{if } f_n[x] \\ g_{n+1}[x] & \text{otherwise.} \end{cases}$$

Example 3. QBF *and* SVR. Let the code ϕ^* for the *quantified boolean formulas* ϕ be defined by ($\beta, \beta_1, \dots$ are *literals*, $\bar{i}$ is i in binary, $OR, EX, \dots$ are atoms) $0^* = T$; $1^* = F$; $\beta_i^* = (VAR, \bar{i})$; $\neg\phi^* = (N, \phi^*)$; $\forall\beta\phi^* = (ALL, \beta^*, \phi^*)$; $\exists\beta\phi^* = (EX, \beta^*, \phi^*)$; $\chi \wedge \psi^* = (AND, \chi^*, \psi^*)$; $\chi \vee \psi^* = (OR, \chi^*, \psi^*)$. Define

$$qb[x; u; z] := \begin{cases} \text{if } z = T, F \text{ then } z \text{ else } val[x; u] & \text{if } x; u; z \text{ minimal} \\ \left|\begin{array}{ll} qb[x; \langle L; u\rangle; (z)_2] \textit{ and } qb[x; \langle R; u\rangle; (z)_3] & \text{if } (z)_1 = AND \\ qb[x; \langle L; u\rangle; (z)_2] \textit{ or } qb[x; \langle R; u\rangle; (z)_3] & \text{if } (z)_1 = OR \\ qb[x; \langle T; u\rangle; (z)_3] \textit{ and } qb[x; \langle F; u\rangle; (z)_3] & \text{if } (z)_1 = ALL \\ qb[x; \langle T; u\rangle; (z)_3] \textit{ or } qb[x; \langle F; u\rangle; (z)_3] & \text{if } (z)_1 = EX \\ not[qb[x; u; (z)_2]] & \text{if } (z)_1 = N \\ qb[x; u; (z)_1] & \text{if } (z)_1 = VAR \end{array}\right. & \text{otherwise.} \end{cases}$$

We now outline the computation implicit in the definition above. x is the *instance* to be decided, and it is the intended initial value for z; u is the *history* of the current value of z. qb parses z, and calls itself recursively, by assigning a subformula to z, and by up-dating the history with: (i) atom $L(R)$ if z begins by a binary connective, and we are choosing the left (right) subformula; (ii) atom T (F) if z begins by a quantifier $Q\beta$, and we are assigning true (false) to β. Assume defined (in the class $\mathscr{PL}$ of next section) a function $val[x; u]$ which: uses the history u to identify an occurrence $\hat{\beta}$ of a literal β; localizes the innermost quantifier $Q\beta$ having $\hat{\beta}$ in its scope; recovers from u the truth-value V assigned to β (and, therefore, to $\hat{\beta}$) when $Q\beta$ was cancelled; and returns V. When the parsing reduces z to a constant or to a literal, qb returns it, or, respectively, uses val to return the truth-value previously assigned to that literal.

The above definition is by SVR, with step function defined by 6 cases, and 10 decreasing tuples. The ith case consists of a boolean function in two ($i \leqslant 4$) auxiliary variables or in one ($i = 5, 6$). The first 8 tuples are given by

$$\mathbf{d}^{ij}[y] = id[y], \langle Z_j; y\rangle, (y)_l,$$

where if $i = 1, 2$ then $Z_j = L, R$; $l = 2, 3$; and where if $i = 3, 4$ then $Z_j = T, F$; $l = 3$. We have $\gamma(\mathbf{d}^{ij}) = -1$ (two destructors and a prefix in the first 8 of them; an id and one destructor in the last two).

Language QBF is accepted (cf. Definition 14) by

$$qbf[x] := qb[x; nil; x],$$

where $nil = (F, F)$ is used (in this and in further constructions) as a *dummy* suffix, preventing inappropriate exits from a recursion, due to atomic values. Accordingly, our functions are tacitly assumed to take nil as a not meaning-conveying entity (this applies for example to function val above).

Example 4. SAT *and* OR-VR. Let us code the sentential formula ϕ by a list x, whose jth component is L, R, AND, OR, NOT, $\bar{i}$ if the jth symbol of ϕ is, respectively, a parenthesis, a connective or literal β_i. Assume defined (in the class $\mathscr{PL}$ of next section) a function $true[x; u]$ which, by input a list of atoms u and x: (a) assigns true (false) to the ith literal of x if the ith component of u is (not) T; (b) returns $T(F)$ if x is true (false) under this truth-assignment. Define by ORVR in $true$ and $h[x; u; \mathbf{z}; \mathbf{s}] = s_1$ *or* s_2, with 4-ples (whose rate of growth is -1) $id[y], \langle V; u\rangle, (\mathbf{z})_2$ $(V = T, F)$

$$st[x; u; z_1; z_2] = \begin{cases} true[x; u] & \text{if } x; u; \mathbf{z} \text{ is minimal,} \\ st[x; \langle T; u\rangle; (\mathbf{z})_2] \text{ or } st[x; \langle F; u\rangle; (\mathbf{z})_2] & \text{otherwise.} \end{cases}$$

Define $sat[x] := st[x; nil; x; x]$.

The computation implicit in this definition is similar to and simpler than the one for QBF. u is a truth-assignment, constructed recursively. At each step, st calls recursively itself twice: in the first call it adds a T to u, in the other it adds an F. Let n be the number of components of x. When the recursion height reaches n, function $true$ is applied to every truth assignment of length $\leqslant n$. The number of distinct literals occurring in the formula coded by x is obviously $\leqslant n$). Thus SAT is accepted (cf. Definition 14) by sat.

4. Characterizations

Notation 12. Given a class of functions $\mathscr{C}$ and a recursion scheme R, we write R($\mathscr{C}$) for the class of all functions definable by a single R in functions in $\mathscr{C}$; and we write $\mathrm{R}^+(\mathscr{C})$ for the closure under R of R($\mathscr{C}$).

Definition 13. Define (cf. Notation 3.6)

PTIME $\mathrm{Lisp}(\mathscr{PL}$, also $\Delta_1^p\mathscr{L}) = \mathrm{FVR}(\mathscr{I})^{\#}$;

PSPACE $\mathrm{Lisp}(\mathscr{PSL}) = \mathrm{SVR}(\mathscr{PL})^{\#}$;

$\Sigma_n^p\mathscr{L} = \mathrm{ORVR}(\Delta_n^p\mathscr{L})^{\#}$;

$\Delta_{n+1}^p\mathscr{L} = \mathrm{FVR}(\Sigma_n^p\mathscr{L})^{\#}$;

PH $\mathrm{Lisp}(\mathscr{PHL}) = \mathrm{ORVR}^+(\mathscr{PL})^{\#}$.

Observe that $f \in \mathscr{C}$ does not imply $not[f[x]] \in \mathscr{C}$. We have $\overline{SAT} \in \Delta_2^p \mathscr{L}$, and $\overline{SAT} \in \mathscr{PHL}$, since we may define by both FVR and ORVR

$$not^*[x] = \begin{cases} not[x] & \text{if } x \text{ is minimal,} \\ F \text{ or } F & \text{otherwise.} \end{cases}$$

Definition 14. $f \in \mathscr{B}$ *accepts* language L if we have $f[\mathbf{x}] = T$ iff $\mathbf{x} \in L$.

Next theorems prove that (1) all classes above are equivalent (in the sense of last definition) to the complexity classes their names suggest; (2) class $\mathscr{PL}$ is closed under FVR, while $\mathscr{PSL}$ is closed under FVR and SVR.

5. Simulation by TMs

Lemma 15. PSPACE *and* PTIME *are respectively closed under* SVR *and* FVR.

Proof. Let $f[\mathbf{y}]$ be defined by SVR (FVR), with q decreasing tuples, r principal variables, and with all other notations like in Definition 10. Let $\theta(\mathbf{y})$ be the q-ary tree: (a) whose root is (labelled by) $\mathbf{y}$; (b) whose leaves are minimal; and (c) whose internal nodes $\mathbf{z}$ have q children $\mathbf{d}^i[\mathbf{z}]$ $(1 \leqslant i \leqslant q)$. Its height is $\leqslant |\mathbf{y}|$.

Case 1: f is defined by SVR with $q \geqslant 1$ or by FVR with $q = 1$.

Simulation: Let g, h, $\mathbf{d}^i$ be simulated by the TMs G, H, $\mathbf{D}^i$. f is simulated by a TM F, using the $\mathbf{D}^i$ to visit $\theta(\mathbf{y})$ in the mode known as *post-order*. It records in a stack Σ_1 the sequence of recursive calls (under the form of *records* containing the current values of the principal variables and the number i of the decreasing tuple yielding them); and it stores in a second stack Σ_2 the values $f[\mathbf{d}^i[\ldots]]$ needed to compute $f[\ldots]$.

Space: In addition to space used by G, H, $\mathbf{D}^i$, F needs space for the stacks; the amount for Σ_1 is quadratic in $|\mathbf{y}|$, since we have to store $\leqslant q|\mathbf{y}|$ records; and since, by Lemma 6, the length of each record is $\leqslant |\mathbf{y}| + \log(q)$. When in Σ_1 there are $m \leqslant q|\mathbf{y}|$ records, in Σ_2 there are $\leqslant qm \leqslant q^2|\mathbf{y}|$ values of f; thus, since these values are atoms, $|\Sigma_2|$ is linear in $|\mathbf{y}|$.

Time: If $q = 1$, then Σ_2 is always empty; $\theta(\mathbf{y})$ has a single path of length $\leqslant |\mathbf{y}|$, visited top-down in a straightforward way. Thus, runtime is polynomial, since G, H and $\mathbf{D}$ are applied less than $|\mathbf{y}|$ times.

Case 2: f is defined by FVR with $q > 1$ tuples consisting of destructors and *id*'s only. Observe that the number of all lists which can be obtained from z by means of destructors is $\leqslant 2|z|$. Hence, for a given r-ple $\mathbf{y}$, the number of the possible values for all the possible $\mathbf{d}^i[\mathbf{y}]$ is $\leqslant 2^r|\mathbf{y}|^r$. We use a TM F^*, which differs from the F above by the following changes. Stack Σ_2 is replaced by a table T in which all couples $\mathbf{z}, f[\mathbf{z}]$ already computed are stored. When F^* visits a node v associated with $\mathbf{z}$, it looks first in T for $\mathbf{z}$ and: (a) if the corresponding value is already in T, it returns it, and moves

to the brother or father of v, ignoring the nodes below it; otherwise (b) it continues to visit $\theta(\mathbf{y})$ in the same way as F, and we have the

Claim. *F^* adds a new couple to T after visiting at most $\leqslant q|\mathbf{z}|$ nodes.*

Proof of the claim. Assume (ad absurdum) not empty the set Γ of all nodes not satisying the claim. Since Γ is finite there exists a $\mathbf{z}\in\Gamma$ which is minimal with respect to the partial order $\prec_\mu$. Observe that $\mathbf{z}$ is not a leaf, and there exists $j\leqslant q$, such that the jth son $\mathbf{u}$ of $\mathbf{z}$ is not in T (since else F^* computes $f[\mathbf{x};\mathbf{z}]$, and adds it to T, after respectively 0 or q visits). But then, since $\mathbf{u}\prec_\mu\mathbf{z}$, we have $\mathbf{u}\notin\Gamma$; hence, F^* adds a new value to T before $j+q|\mathbf{u}|\leqslant q|\mathbf{z}|$ visits.

Time is now polynomial in $\mathbf{y}$, since F^* applies G, H, $\mathbf{D}^i$ for less than $2^r|\mathbf{y}|^r$ times; since, by the claim, filling T requires at most $q|\mathbf{y}|\cdot 2^r|\mathbf{y}|^r$ searches through a table of length $\leqslant|\mathbf{y}|2^r|\mathbf{y}|^r$; and since, when T is full, it returns $f[\mathbf{y}]$. □

Theorem 16. *We have $\mathscr{PL}\subseteq$ PTIME, and $\mathscr{PSL}\subseteq$ PSPACE.*

Proof. By last lemma, by closure of PTIME and PSPACE under substitution, and since all functions in $\mathscr{I}$ can be simulated in DTIME(n^2). □

6. Simulation of TMs

TMs, *tapes*, *codes*. We restrict ourselves to tapes and (not empty) words X over alphabet $\{0,1\}$. The code x for X is the simple list whose ith component is 0(1) iff the i-th last bit of X is 0(1). Thus, since we always have $\#(x)=|x|=|X|$, we may identify words and tapes with their codes. This allows saying, for example, that a TM, by input $\mathbf{x}\in L$, stops operating within time $T(|\mathbf{x}|)$. In addition to the semi-tape and to the $p+1$ *ordinary states* $0,\ldots,p$ of ordinary TMs, an oracle TM has: (a) 3 oracle states $p+1,\ldots,p+3$); and (b) a push-down *oracle tape* O. When in state $p+1$ ($p+2$), an oracle TM pushes the observed symbol (kills the top symbol), and enters state p. When in state $p+3$, it enters state $p-1$ ($p-2$) if O is (not) accepted by the oracle. All TMs, by input x on their first $|x|$ cells, start operating in state 1 over a 0 in cell $|x|+1$, with O empty; they *accept* by writing a 0 in some cell, and by entering, in state 0, an endless loop over that cell.

We use 1 and 0 for the tape symbols, plus $2(p+4)$ atoms X_j^i. An atom in this form represents the *core* of an istantaneous description, in the sense that it says that M scans a j and is in state i.

Given M, $P_{jj^*}^{ii^*}$ and $P_{j,\pm}^{ii^*}$ ($i\leqslant p$; $i^*\leqslant p+3$; j, $j^*\leqslant 1$) are predicates, which are true iff, when X_j^i is the current core, (their argument) M enters state i^*, and, respectively, writes j^*, moves right, left. P_{00}^{00} ensures that, when M accepts, the cores of its id's reduce to X_0^0.

M^A is a TM with oracle A, and c_A is a boolean function accepting A.

Lemma 17. *For every M^A there exists function $sim_{M^A} \in \text{FVR}(\mathscr{I} \cup \{c_A\})$ accepting x and a list y whose components are all* 0, *iff M^A accepts x within $|y|$ steps.*

Proof. Let the *boolean-ization* $\tilde{f}$ of a given f be given by $\tilde{f}[x] := cond[f[x]; T; F]$. Define (one line for each system of values for i, i^*, j, j^*, l, r, defined as above)

$$
sm[w_1; w_2; w_3; z; u_1; u_2] :=
$$

$$
\begin{cases}
\left|\begin{array}{ll} T & \text{if } (z)_1 = X_0^0 \\ F & \text{otherwise} \end{array}\right. & \text{if } \mathbf{w}; z; \mathbf{u} \text{ is minimal; else:} \\
\widetilde{sm}[\mathbf{w}; \langle X_{j^*}^{i^*}; \mathrm{T}z\rangle; \mathbf{Tu}] & \text{if } (z)_1 = X_j^i \text{ and } P_{jj^*}^{ii^*} \\
\widetilde{sm}[\langle j; w_1\rangle; \mathrm{T}\ w_2; w_3; \langle X_r^{i^*}; \mathrm{T}z\rangle; \mathbf{Tu}] & \text{if } adj_{lr}[w_1; w_2], (z)_1 = X_j^i \text{ and } P_{j,+}^{ii^*} \\
\widetilde{sm}[\mathrm{T}\ w_1; \langle j; w_2\rangle; w_3; \langle X_l^{i^*}; \mathrm{T}z\rangle; \mathbf{Tu}] & \text{if } adj_{lr}[w_1; w_2], (z)_1 = X_j^i \text{ and } P_{j,-}^{ii^*} \\
\widetilde{sm}[w_1; w_2; \langle j; w_3\rangle; \langle X_j^p;\ \mathrm{T}z\rangle; \mathbf{Tu}] & \text{if } (z)_1 = X_j^{p+1} \\
\widetilde{sm}[w_1; w_2; \mathrm{T}\ w_3; \langle X_j^p; \mathrm{T}z\rangle; \mathbf{Tu}] & \text{if } (z)_1 = X_j^{p+2} \\
\left|\begin{array}{l} \widetilde{sm}[\mathbf{w}; \langle X_j^{p-1}; \mathrm{T}z\rangle; \mathbf{Tu}] \text{ if } c_A[w_3] = T \\ \widetilde{sm}[\mathbf{w}; \langle X_j^{p-2}; \mathrm{T}z\rangle; \mathbf{Tu}] \text{ if } c_A[w_3] \neq T \end{array}\right. & \text{if } (z)_1 = X_j^{p+3}
\end{cases}
$$

where $adj_{lr}[w_1; w_2] := \mathrm{H}w_1 = l$ and $\mathrm{T}w_2 = r$.

Define further $sim_{M^A}[x; y] := sm[x; y; nil; \langle X_0^1; nil\rangle; app[y; F]; app[y; F]]]$.

Variable w_1 gives the part of the tape, at the left of o, read backward (so that its first atom gives the symbol at the left of o); w_2 and w_3 give the part at the right and O, read in the usual way; thus functions adj_{lr} are accepting iff l, r are the symbols adjacent to o. We have $z = (X, nil)$ if X is the current core. The u's are counters. Assume that, by step $t < |y|$, M^A is in state $i \leqslant p$, scans a j, and that the cells at the left and right of o contain, say, a 0 and a 1, respectively. Assume further that we have $P_{j,+}^{ii^*}$, and that, therefore, M^A has to move right. The line corresponding to $l = 0,\ r = 1$ applies. It says that now $o = 1$, the tail of w_2 is at the right of o, while j is the head of the part of tape now at its left. Assume that we are visiting o for the first time; since $t < |y|$ and since y consists of zeroes, we have $o = 0$.

By scanning the above definition, we see that all functions used to construct *sm*, but c_A, are in $\mathscr{I}$. To see that *sm* is FVR in these functions, observe that the rate of growth of all tuples used for the recursive calls is $\leqslant -1$ (3 or 2 destructors versus 2 or 1 prefix).

The lemma follows by observing that if y is simple, then its second and third occurrences among the arguments of sim_{M^A} reduce to an atom after $|y|$ recursive steps, and that the first copy of y ensures room enough to move right $|y|$ times. □

Lemma 18. *For every j there exists $f_j^{\pi}[x] \in \mathscr{I}$, which associates each simple list with a list of length $\geqslant j|x|^j + j$, whose components are all* 0.

Proof. Define $f_j^{\pi}[x] := clone[0; clone[x; x]]$. If x is simple we have that $|f_j^{\pi}[x]| = |x|^2$. The result follows, since addition in unary is ensured by function *app*, and since $\mathscr{I}$ is closed under substitution. □

Theorem 19. *We have* PTIME $\subseteq \mathscr{PL}$.

Proof. Let language L be given, together with an instance X, coded by x. Let L be accepted within time $jn^j + j$ by the ordinary TM M respecting the conventions at the beginning of this section. By last two lemmas, L is accepted by $sim_M[x; f_j^{\pi}[x]]$. This function is in FVR$(\mathscr{I})^{\#}$, since it is defined by substitution of a function in $\mathscr{I}$ inside a function $\in$ FVR$(\mathscr{I})$ (since c_A is absent). □

Theorem 20. $\mathscr{PL}$ *is closed under* FVR.

Proof. By Theorem 19, since, by Lemma 15, PTIME is closed under this scheme. □

Theorem 21. *We have* PSPACE $\subseteq \mathscr{PSL}$.

Proof. Let language $L \in$ PSPACE be given, together with a function ϕ reducing L to QBF. Code (the value of) $\phi(x)$ and define the history u of a subformula of ϕ like in Example 3. Let us denote by ψ_u the formula whose history is u, and by Z_{xu} its outermost atom. We can define the following poly-time TMs:

(a) M_{prs}, which is a parser, taking x, u into Z_{xu}; for technical reasons we find expedient to assume that M_{prs}, when a literal is reached, stops its parsing to enter an invariant cycle always returning atom *VAR*;
(b) M_{ϕ}, which computes $\phi(x)$ within time $n|x|^n + n$, for some n; and $M_{\text{prs},\phi}$ which, by input x, u, yields the output of M_{prs} by input $\phi(x), u$;
(c) M_{vl} which, by input x, u, returns T (F) if u is the history of a true (false) occurrence $\hat{\beta}$ of literal β in $\phi(x)$.

By Theorem 19 we can define in $\mathscr{PL}$ the functions $main[x; u] = Z_{xu}$ and $val^*[x; u]$ simulating the TMs $M_{\text{prs}}(M_{\phi}(x), u)$ and M_{vl}. Define $g[x; u; z_1; z_2] =$

$$
\begin{cases}
\text{if } z = T, F \text{ then } z \text{ else } val[x; u] & \text{if } x; u; z \text{ minimal; else:} \\
\quad g[\mathbf{d}^{11}[x; u; \mathbf{z}]] \text{ and } g[\mathbf{d}^{12}[x; u; \mathbf{z}]] & \text{if } main[x; u] = AND \\
\quad g[\mathbf{d}^{41}[x; u; \mathbf{z}]] \text{ or } g[\mathbf{d}^{42}[x; u; \mathbf{z}]] & \text{if } main[x; u] = EX \\
\quad \text{similarly} & \text{if } main[x; u] = OR, ALL \\
\quad not[g[x; u; \mathrm{T}\mathbf{z}]] & \text{if } main[x; u] = N \\
\quad g[x; u; \mathrm{T}\mathbf{z}] & \text{if } main[x; u] = VAR,
\end{cases}
$$

where, for example, $\mathbf{d}^{11}[v] = id, \langle L; v\rangle, \mathrm{T}v, \mathrm{T}v$, $\mathbf{d}^{42}[v] = id, \langle F; v\rangle, \mathrm{T}v, \mathrm{T}v$. Define further (cf. Lemma 18)

$$f[x] = g[x; nil; f_n^{\pi}[x]; f_n^{\pi}[x]].$$

By an input in this form, g behaves in the following way. While z_1 is not an atom (that is for $n|x|^n + n$ times) and while ψ_u is not a literal, it queries Z_{xu} to *main*. Assume for example $main[x; u] = EX$. We then have

$$g[x; u; \mathbf{z}] = g[x; \langle T; u\rangle; \mathrm{T}\mathbf{z}] \quad \text{or} \quad g[x; \langle F; u\rangle; \mathrm{T}\mathbf{z}].$$

When a literal is found, M_{prs} starts returning identically *VAR*, and g enters an invariant cycle which leaves un-changed the history. When z_1 reduces to an atom, a truth-value is returned.

g is defined by SVR in $val^*, main \in \mathscr{PSL}$ and is, therefore, in $\mathscr{PSSL}$. The result follows, since f is defined by substitution of functions in $\mathscr{I}$ inside functions in $\mathscr{PSSL}$.

Note 22. This proof is crucial for our choice of a tree-algebra. Other proofs in this paper can be adapted to a word algebra having a *pairing* and two *unpairing* functions among its initial functions. However, if pairing is defined via concatenation (see the ternary notations used by Schwichtenberg [18]), un-pairing is difficult with functions whose ranges reduce to atoms: we did not succeed in accepting QBF with un-nested analogues for notations of SVR. If codes are defined by means of triangular numbers, instead of word-algebraic constructors, then reduction of $L \in$ PSPACE to QBF requires an exponential space, which is incompatible with the proof of last theorem.

Theorem 23. *$\mathscr{PSL}$ is closed under* FVR *and* SVR.

Proof. By Theorem 21, since by Lemma 15 PSPACE is closed under these schemes. □

7. Equivalence of the two hierarchies

Theorem 24. (1) *For all n, Σ_n^p and Δ_n^p are resp. equivalent to $\Sigma_n^p\mathscr{L}$ and $\Delta_n^p\mathscr{L}$.*
(2) *The polynomial hierarchy and $\mathscr{PHL}$ are equivalent.*

Proof. (1) Induction on n. We have to show four points.

(i) $\Delta_n^p \subseteq \Delta_n^p\mathscr{L}$. Basis. By Theorem 16. Step. Let L be accepted in polynomial time by M^{L^*}, with $L^* \in \Sigma_n^p$. By the ind. hyp. c_{L^*} is definable in $\Sigma_n^p\mathscr{L}$. The result follows by Lemma 17, by arguments like in proof of Theorem 19.

(ii) $\Sigma_n^p \subseteq \Sigma_n^p\mathscr{L}$. Let $L \in \Sigma_n^p$ be given. There exist j and $L^* \in \Delta_n^p$ such that

$$x \in L \Leftrightarrow \exists ywz(|y| \leqslant j|x|^j + j \text{ and } y = \langle w; z\rangle \text{ and words } x, w \text{ belong to } L^*).$$

By part (i) of this proof, c_{L^*} is definable in $\Delta_n^p\mathscr{L}$. Define

$$f[x;y;u_1;u_2]=\begin{cases} c_{L^*}[x;y] & \text{if } x;y;\mathbf{u} \text{ is minimal,}\\ f[x;\langle 1;y\rangle;\mathbf{Tu}] \text{ or } f[x;\langle 0;y\rangle;\mathbf{Tu}] & \text{otherwise.}\end{cases}$$

The above definition is by ORVR in c_{L^*}, by reasons repeteadly used in previous proofs and examples. We then have $c_L[x]=f[x;nil;f_j^\pi[x];f_j^\pi[x]]$.

(iii) $\Sigma_n^p\mathscr{L}\subseteq\Sigma_n^p$. Let $f[\mathbf{x};\mathbf{y}]$ be defined by ORVR in $c_L\in\Delta_n^p\mathscr{L}$ and h, with decreasing tuples $\mathbf{d}^{ij}$. A nondeterministic TM M_f^L can be defined, which: (a) iterates an invariant cycle, including, at each *or* of h_i, the choice of a j, and the simulation of $\mathbf{d}^{ij}$; (b) at each call to c_L, queries the oracle. Runtime for M_f^L is polynomial ($\leqslant|\mathbf{y}|$ simulations of functions $\mathbf{d}^{ij}$, and one application of c_L for each nondeterministic computation).

(iv) $\Delta_n^p\mathscr{L}\subseteq\Delta_n^p$. Closure of Δ_n^p under definitions by FVR can be proved by the same arguments as in proof of Lemma 15.

(2) $\mathscr{PHL}\subseteq$ PH follows by part (1). To prove the other half, let $L\in$ PH be given. There exist a relation $R(\mathbf{y},x,z)$, a polytime boolean function ϕ and a j such that we have

$$R(\mathbf{y},x,z)\ \Leftrightarrow\ Q_1(y_1)\leqslant z\ldots Q_m(y_m)\leqslant z(\phi(\mathbf{y},x)),\ Q_i=\exists,\forall,\ Q_i\neq Q_{i+1},$$

$$x\in L\ \Leftrightarrow\ R(\mathbf{y},x,j|x|^j+j).$$

Let $c_R[x;z]$ be the characteristic function of R. We have that c_R is definable in $\mathscr{PHL}$ (by part (1) if $Q_1=\exists$; by part (1) and by recalling that negation can be defined by ORVR if $Q_1=\forall$). The result follows by defining (by substitution of a function in $\mathscr{I}$ inside a function in $\mathscr{PHL}$)

$$c_L[x]=c_R[x;f_j^\pi[x]].$$

Note 25. Two atoms only are sufficient to prove our characterizations. To this purpose, in proofs of Lemma 17, and Theorems 21 and 24 replace: (1) all atoms Z like X_j^i or *AND*, *VAL*,... by simple lists x_Z of a pre-assigned length n (depending on the simulated TM, in the former case); (2) some destructors and prefixes in the decreasing tuples by sequences of n destructors and prefixes — this does not affect the rates of growth; (3) some tests of the form $y=Z$ by sequences of n alternate cond's and destructors deciding test $y=x_Z$.

8. Conclusion

A main difference between variants of VR is the adopted order. For reasons mentioned in the Introduction, *full* VR (that is, with an order isomorphic to ordinary $<$

on natural numbers) is unfeasible. Stricter orders have, therefore, to be selected. If *comparability* and *naturalness* have to be the main criteria, then the obvious candidate is $x \prec_\mu y =_{df} |x| < |y|$. If applications have to be considered too (and VR may be regarded as a mathematical explanation of definability of recursive procedures), then "*x occurs in y*" qualifies as adequate sub-order of $\prec_\mu$. If *parsing* is a worth refinement of mere *occurrence*, then a tree-algebra better copes with data's very structure than the kind of coding tricks more or less unavoidable with word algebras. Add that certain asymmetries may disturb the simultaneous treatment of notations with $\prec_\mu$ and $\prec_{occ}$, since, for example, we may have $|2^{x/2}3^{y/2}5^{2z}| > |2^x 3^y 5^z|$.

A first validation of our vote in favour of trees (and Lisp) may come by comparing our simple-minded proofs with, for example, the difficulty of those in [3], or in the implementation of Cobham's ideas in [15], p. 126. Or it may come by scanning [10, Sections 51, 57]: all algorithms for the first Gödel theorem and for predicate T (a universal function) are written in a language quite close to $\mathscr{PL}$ (the only essential change is replacing all bounded quantifications by an easy FVR definition of a function for search of sub-expressions).

References

[1] S.J. Bellantoni, Predicative recursion and computational complexity, Ph.D. Thesis, Toronto, 1992.

[2] S.J. Bellantoni, Predicative recursion and the polytime hierarchy, in: P. Clote, J. Remmel (Eds.), Feasible Mathematics II, Birkhäuser, Basel, 1994.

[3] S. Bellantoni, S. Cook, A new recursion-theoretic characterization of the poly-time functions, Comput. Complexity 2 (1992) 97–110.

[4] S.J. Bellantoni, K.-H. Niggl, Ranking primitive recursion: the low Grzegorczyck classes revisited, 1997, submitted for publication.

[5] S. Caporaso, Safe Turing machines, Grzegorczyk classes and Polytime, Int. J. Found. Comp. Sci. 7.3 (1996) 241–252.

[6] S. Caporaso, G. Pani, Undecidability vs transfinite induction for the consistency of hyperarithmetical sets, Arch. Math. Logik Grundig. 22 (1982) 19–26.

[7] S. Caporaso, M. Zito, On a relation between uniform coding and problems of the form DTIME(F) = ? DSPACE(F), Acta Inform. 35 (1998) 1–8.

[8] A. Cobham, The intrinsic computational difficulty of functions, in: Y. Bar Hillel (Ed.), Proc. Int. Conf. Logic, Methodology and Philosophy Sci., North-Holland, Amsterdam, 1965, pp. 24–30.

[9] S. Fefermann, Systems of predicative analysis I and II, JSL 29 (1964) 1–30 and JSL 33 (1968) 193–200.

[10] S.C. Kleene, Introduction to Metamathematics, North-Holland, Amsterdam, 1952.

[11] D. Leivant, A foundational delineation of computational feasibility, Proc. 6th Annual IEEE Symp. Logic in Computer Science, IEEE Computer Society Press, Silver spring, MD, 1991.

[12] D. Leivant, Ramified recurrence and computational complexity I: word recurrence and polytime, in: P. Clote, J. Remmel (Eds.), Feasible Mathematics II, Birkhäuser, Basel, 1994.

[13] D. Leivant, J.-Y. Marion, Ramified recurrence and computational complexity II: substitution and polyspace, in: J. Tiuryn, L. Pacholski (Eds.), Computer Science Logic, Lecture Notes in Computer Science, Vol. 133, Springer, Berlin, 1994, pp. 486–500.

[14] H. Poincaré, Les mathématiques et la logique, Rev. Métaphis. Morale 14 (1906) 297–317.

[15] H.E. Rose, Subrecursion: Functions and Hierarchies, Oxford Press, Oxford, 1984.

[16] J.R. Schoenfield, Axioms of set theory, in: J. Barwise (Ed.), Handbook of Mathematical Logic, North-Holland, Amsterdam, 1977.
[17] K. Schütte, Beweistheoretische Untersuchung der Verweigten Analysis, Math. Ann. 124 (1952) 123–147.
[18] H. Schwichtenberg, Rekursionszahlen und die Grzegorczyk-Hierarchie, Arch. Math. Logik Grundig 12 (1969) 61–74.