Computation Models and Function Algebras

P. Clote*

Contents

1 Introduction **2**

2 Machine Models **4**

2.1 Turing machines 4

2.2 Parallel machine model 11

2.3 Circuit families 14

3 Some recursion schemes **16**

3.1 An algebra for the logtime hierarchy LH 17

3.2 Bounded recursion on notation 27

3.3 Bounded recursion 29

3.4 Bounded minimization 35

3.5 Divide and conquer, course-of-values and miscellaneous 39

3.6 Safe recursion 45

4 Type 2 functionals **52**

5 Acknowledgements **63**

*A preliminary, abbreviated version of this paper appears in the proceedings of *Logic and Computational Complexity*, edited by D. Leivant, *Springer Lecture Notes for Computer Science* **960** 98–130 (1995). This research partially supported by NSF CCR-9408090, US – Czechoslovak Science and Technology Program Grant 93-025 and the Volkswagen Foundation.
Address: Institut für Informatik, Ludwig-Maximilians-Universität München, Oettingenstr. 67, D-80538 München. `clote@informatik.uni-muenchen.de`

1 Introduction

The modern digital computer, a force which has shaped the latter part of the 20-th century, can trace its origins back to work in mathematical logic concerning the formalization of concepts such as *proof* and *computable function*. Numerous examples support this assertion. For instance, in his development of the universal Turing machine, A.M. Turing seems to have been the first, along with J. von Neumann, to have understood the potential of memory-stored programs executed by a universal computational device. Moreover, certain function classes and proof systems can be viewed as prototypes of programming languages: LISP was developed from the Church-Kleene λ-calculus; PROLOG was developed from resolution (Gentzen sequent calculus); polymorphic programming languages such as ML were inspired by J.-Y. Girard's system **F**; imperative programming languages such as PASCAL and C can be viewed as an implementation of S.C. Kleene's μ-recursive functions.

One recurring theme in recursion theory is that of a *function algebra* — i.e. a smallest class of functions containing certain initial functions and closed under certain operations (especially substitution and primitive recursion).[1] In 1904, G.H. Hardy [66] used related concepts to define sets of real numbers of cardinality $\aleph_1$. In 1923, Th. Skolem [131] introduced the primitive recursive functions, and in 1925, as a technical tool in his claimed sketch proof of the continuum hypothesis, D. Hilbert [70] defined classes of higher type functionals by recursion. In 1928, W. Ackermann [1] furnished a proof that the diagonal function $\varphi_a(a,a)$ of Hilbert [70], a variant of the Ackermann function, is not primitive recursive. In 1931, K. Gödel [53] defined the primitive recursive functions, there calling them "rekursive Funktionen", and used them to arithmetize logical syntax via Gödel numbers for his incompleteness theorem. Generalizing Ackermann's work, in 1936 R. Péter [111] defined and studied the k-fold recursive functions. The same year saw the introduction of the fundamental concepts of Turing machine (A.M. Turing [137]), λ-calculus (A. Church [26]) and μ-recursive functions (S.C. Kleene [82]). By restricting the scheme of primitive recursion to allow only limited summations and limited products, the *elementary functions* were introduced in 1943 by L. Kalmár [78]. In 1953, A. Grzegorczyk [58] studied the classes $\mathcal{E}^k$ obtained by closing certain fast growing "diagonal" functions under composition and *bounded primitive recursion* or *bounded minimization*.

H. Scholz's 1952 [122] question concerning the characterization of *spectra* $\{n \in \mathbf{N} : (\exists \text{ model } M \text{ of } n \text{ elements})(M \models \phi)\}$ of first order sentences ϕ, which was shown in 1974 by N. Jones and A. Selman [77] to equal $\text{NTIME}(2^{O(n)})$, was the starting point for J.H. Bennett's work [13] in 1962. Among other results, Bennett introduced the key notions of *positive extended rudimentary* and *extended rudimentary* (equivalent to the notions of nondeterministic polynomial time NP and the polynomial time hierarchy PH), characterized the spectra of sentences of higher type logic as exactly the Kalmár elementary sets, and proved that *rudimentary* coincides with Smullyan's notion of *constructive arithmetic* (those sets definable in the language $\{0,1,+,\cdot,\leq\}$ of arithmetic by first order bounded quantifier formulas). Only much later in 1976 did C. Wrathall [145] connect these concepts to computer science by proving that the linear time hierarchy LTH coincides with rudimentary, hence constructive arithmetic, sets. In 1963 R. W. Ritchie [114] proved that Grzegorczyk's class $\mathcal{E}^2$ is the collection of functions computable in linear space on a Turing machine. In 1965, A. Cobham [40] characterized the polynomial time computable functions as the smallest function algebra closed under Bennett's scheme

[1] In [70], Hilbert stated that "*substitution* (i.e. replacement of an argument by a new variable or function) and *recursion* (the scheme of deriving the function value for $n+1$ from that of n)" are "the elementary operations for the construction of functions".

of *bounded recursion on notation.* [2] These arithmetization techniques led to a host of characterizations of computational complexity classes by machine-independent function algebras in the work of D. B. Thompson [135] in 1972 on polynomial space, of K. Wagner [141] in 1979 on general time complexity classes. Function algebra characterizations of *parallel* complexity classes were given more recently by the author [39] in 1990 and B. Allen [3] in 1991, while certain small *boolean circuit* complexity classes were treated by the author and G. Takeuti [37] in 1995. Higher type analogues of certain characterizations were given in 1976 by K. Mehlhorn [96], in 1991 by S. Cook and B. Kapron [80, 44] for sequential computation, and in 1993 by the author, A. Ignjatovic, B. Kapron [34] for parallel computation. In 1995 H. Vollmer and K. Wagner [140] Valiant's class $\#P$. Though distinct, the arithmetization techniques of function algebras are related to those used in proving numerous results like *(i)* NP equals generalized first order spectra (R. Fagin [48]), *(ii)* the characterization of complexity classes via finite models (the program of *descriptive complexity theory* investigated by R. Fagin [49], N. Immerman [73, 74], Y. Gurevich and S. Shelah [60], and others).

From this short historical overview, it clearly emerges that *function algebras* and *computation models* are intimately related as the software (class of programs) and hardware (machine model) counterparts of each other. Historically, these notions are among the central concepts of recursion theory, proof theory and theoretical computer science. Perhaps this is the reason that K. Gödel [54] claimed in 1975 that the most important open problem in recursion theory is the classification of all total recursive functions, presumably in a hierarchy of function algebras determined by admitting more and more complex operations. While much work characterizing ever larger subrecursive hierarchies has been done by W. Buchholz, J.-Y. Girard, G.E. Sacks, K. Schütte, H. Schwichtenberg, G. Takeuti, S.S. Wainer and others, in this paper we concentrate principally on subclasses of the primitive recursive functions and their relations to computational complexity. For primitive recursive functions, ϵ_0-functions, etc. and strong higher type functionals, see the articles of H. Schwichtenberg and D. Normann in this volume.

Apart from its interest as part of recursion theory, there are applications of function algebras to proof theory, especially in the study of theories T of first and second order arithmetic, whose *provably total* functions (having suitably definable graphs) coincide with those of a particular function algebra. Using such techniques, for instance, in [134] G. Takeuti provided a simpler proof of the existence of an alternating logtime algorithm for the boolean formula evaluation problem, a result first proved by S. Buss [20, 22] (see Theorem 2.11). For a further discussion of such applications, see the recent monograph by J. Krajíček [85].

Historically, Cobham's machine independent characterization of the polynomial time computable functions was the start of modern complexity theory, indicating a robust and mathematically interesting field. As outlined in section 4, current work on type 2 and higher type function algebras suggests directions for the extension of complexity theory to higher type computation. The development of function algebras is potentially important in computer science for programming language design. New kinds of operations used in defining function algebras could possibly be incorporated in *small*, non-universal programming languages for dedicated purposes. All the function algebras defined in this paper could be used to define free variable equational calculi. For instance, S. Cook's system PV [43] comes from Theorem 3.19, the author's systems AV, ALV, ALV' [28, 30] come from Theorems 3.26 and 3.27, J. Johannsen's [75] systems $TV, A2V$ come from Theorem 3.16, while M. O'Donnell [105] has proposed equational calculus as a programming language.

In this paper, we will survey a selection of results which illustrate the arithme-

[2] According to [96], K. Weihrauch independently proved a similar characterization in 1972.

tization techniques used in characterizing certain computation models by function algebras.

2 Machine Models

Despite the immense diversity of abstract machine models and complexity classes (see for instance [139] or [142]), only the most natural and robust models and classes will be treated in this paper. Many of the following machine models are familiar. Nevertheless, definitions are given in sufficient detail to provide an idea of the required initial functions and closure operations which permit function algebra characterizations of complexity classes.

2.1 Turing machines

In proving the recursive unsolvability of Hilbert's *Entscheidungsproblem* (independently established as well by A. Church [26] using the λ-calculus), A.M. Turing [137] introduced the Turing machine, largely motivated by his attempt to make precise the notion of computable (real) number, i.e., "those whose decimals which are calculable by finite means". Considering the "computer" as an idealized human clerk, Turing argued that the "behavior of the computer at any moment is determined by the symbols which he is observing, and his 'state of mind' at that moment", and specified that the number of "states of mind" should be finite, since "human memory is necessarily limited". Formally, we have the following.

Definition 2.1 A multitape Turing machine (TM) M is specified by $(Q, \Sigma, \Gamma, \delta, q_0, k)$ where $k \in \mathbf{N}$,

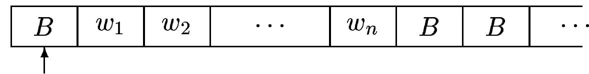
- Q is a finite set of *states* containing the accept and reject states q_A, q_R, as well as the start state q_0,
- Σ [resp. Γ] is a finite read-only input [resp. read-write work] tape alphabet not containing the blank symbol B,
- δ is the transition function and maps

$$(Q - \{q_A, q_R\}) \times (\Sigma \cup \{B\}) \times (\Gamma \cup \{B\})^k$$

  into

$$Q \times (\Gamma \cup \{B\})^k \times \{-1, 0, 1\}^{k+1}.$$

A Turing machine is assumed to have a one-way infinite input tape and k one-way infinite work tapes. The work tapes are initially blank, while on input $w = w_1 \cdots w_n$ with $w_i \in \Sigma$, the initial input tape is of the form below.



Each work tape has a tape head (above indicated by an arrow) capable of reading the symbol in the currently scanned square, writing a symbol in that square and remaining stationary or moving one square left or right. The leftmost cell is the 0-th cell. Since the input tape is read-only, the input tape head can scan a tape cell and remain stationary or move one square left or right. A *configuration* is a member of $Q \times (\Sigma \cup \{B\})^* \times (\Gamma \cup \{B\})^{*k} \times \mathbf{N}^{k+1}$, and indicates the current state, tape contents, and head positions. Alternately, a configuration can be abbreviated by underscoring the symbols currently scanned by a tape head, in order to indicate

the current tape head position. For instance, $(q, Bab\underline{a}B, Bb\underline{b}B)$ abbreviates the configuration of a TM in state q, with an input tape, whose head currently scans an a, and one work tape, whose head currently scans a b. A halted configuration is one whose state is q_A or q_R.

Let

$$\begin{aligned} \alpha &= (q, BxB, \alpha_1, \ldots, \alpha_k, n_0, n_1, \ldots, n_k) \\ \beta &= (r, BxB, \beta_1, \ldots, \beta_k, m_0, m_1, \ldots, m_k) \end{aligned}$$

be configurations for M on input x. Then β is the *next configuration* after α in M's computation on x, denoted $\alpha \vdash_M \beta$, if the following conditions are satisfied:

1. the n_0-th cell of the input tape BxB contains symbol a,
2. for $1 \leq i \leq k$ the following hold:
   (a) $\sigma_i, \tau_i \in \Gamma \cup \{B\}$ and $u_i, v_i, w_i \in (\Gamma \cup \{B\})^*$
   (b) $\alpha_i = u_i \sigma_i v_i$ and $\beta_i = u_i \tau_i w_i$
   (c) $|u_i| = n_i$ (Recall that the leftmost cell is the 0-th cell, so the n-th cell has n cells to its left. This implies that σ_i [resp. τ_i] is the contents of the n_i-th cell of the i-th tape in configuration α [resp. β].)
3. $\delta(q, a, \sigma_1, \ldots, \sigma_k) = (r, \tau_1, \ldots, \tau_k, m_0 - n_0, m_1 - n_1, \ldots, m_k - n_k)$, where for $1 \leq i \leq k$:
   (a) $m_i < |\beta_i|$
   (b) either $v_i = w_i$ or $v_i = \lambda$ (the empty word), $w_i = B$, and $m_i = n_i + 1$.

The reflexive, transitive closure of $\vdash_M$ is denoted by $\vdash_M^*$, and a configuration C is said to *yield* a configuration D *in n-steps*, denoted $C \vdash_M^n D$, if there are $C_1, \ldots, C_n$ such that $C = C_1 \vdash_M C_2 \vdash_M \cdots \vdash_M C_n = D$, while C *yields* D if $C \vdash_M^* D$. A Turing machine M *accepts* a language $L \subseteq \Sigma^*$, denoted by $L = L(M)$, if L is the collection of words w such that the *initial configuration* $(q_0, \underline{B}wB, \underline{B}, \ldots, \underline{B})$ yields $(q_A, \underline{B}wB, \underline{B}, \ldots, \underline{B})$; a word w is *accepted* in n steps if $(q_0, \underline{B}wB, \underline{B}, \ldots, \underline{B}) \vdash_M^n (q_A, \underline{B}wB, \underline{B}, \ldots, \underline{B})$. The machine M accepts $L \subseteq \Sigma^*$ in *time* $T(n)$ (resp. *space* $S(n)$) if $L = L(M)$ and for each word $w \in L(M)$ of length n, w is accepted in at most $T(n)$ steps (resp. the maximum number of cells visited on each of M's work tapes is $S(n)$). A language $L \subseteq \Sigma^*$ is *decided* by M in *time* $T(n)$ (resp. *space* $S(n)$) if L [resp. $\Sigma^* - L$] is the collection of words for which M halts in state q_A [resp. q_R], and for each word $w \in \Sigma^*$ of length n, M halts in at most $T(n)$ steps (resp. the maximum number of cells visited on each of M's work tapes is $S(n)$). This article concerns complexity classes, so for the most part we identify the notions of acceptance and decision (for most of the complexity classes here considered, machines of a certain complexity class can be clocked so as to reject a word if they don't accept it).

Recall that

$$\begin{aligned} O(f) &= \{g : (\exists c > 0)(\exists n_0)(\forall n \geq n_0)[g(n) \leq c \cdot f(n)]\}, \\ \Omega(f) &= \{g : (\exists c > 0)(\exists n_0)(\forall n \geq n_0)[f(n) \leq c \cdot g(n)]\} \\ \Theta(f) &= O(f) \cap \Omega(f) \end{aligned}$$

so that $n^{O(1)}$ denotes the set of all polynomially bounded functions. If T, S are one-place functions, then

$$\begin{aligned}
\text{DTIME}(T(n)) &= \{L \subseteq \Sigma^* : L \text{ accepted by a TM in time } O(T(n))\} \\
\text{DSPACE}(S(n)) &= \{L \subseteq \Sigma^* : L \text{ accepted by a TM in space } O(S(n))\} \\
\text{PTIME} &= \text{P} = \text{DTIME}(n^{O(1)}) \\
\text{PSPACE} &= \text{DSPACE}(n^{O(1)}) \\
\text{ETIME} &= \cup_{c\geq 1}\text{DTIME}(2^{c\cdot n}) = \text{DTIME}(2^{O(n)}) \\
\text{EXPTIME} &= \cup_{c\geq 1}\text{DTIME}(2^{n^c}) = \text{DTIME}(2^{n^{O(1)}}).
\end{aligned}$$

Finally, DTimeSpace$(T(n), S(n))$ is defined as

$$\{L \subseteq \Sigma^* : L \text{ accepted by a TM in time } O(T(n)) \text{ and space } O(S(n))\}.$$

Definition 2.2 A nondeterministic multitape Turing machine (NTM) M is specified by $(Q, \Sigma, \Gamma, \Delta, q_0, k)$ where $Q, \Sigma, \Gamma, q_0, k$ are as in Definition 2.1 and the *transition relation* Δ is contained in

$$\left((Q - \{q_A, q_R\}) \times (\Sigma \cup \{B\}) \times (\Gamma \cup \{B\})^k\right) \times \left(Q \times (\Gamma \cup \{B\})^k \times \{-1, 0, 1\}^{k+1}\right).$$

If α, β are configurations in the computation of the nondeterministic Turing machine (NTM) M on input x, then write $\alpha \vdash_M \beta$ if

$$(q, a, \sigma_1, \ldots, \sigma_k, r, \tau_1, \ldots, \tau_k, m_0 - n_0, m_1 - n_1, \ldots, m_k - n_k) \in \Delta,$$

where $\sigma_i, \tau_i, a, n_i, m_i$ are as in the deterministic case.

With this change, the notions of configuration, yield and acceptance are analogous to the previously defined notions. A nondeterministic computation corresponds to a *computational tree* whose root is the initial configuration, whose leaves are halted computations, and whose internal nodes α have as children those configurations β obtained in one step from α, $\alpha \vdash_M \beta$. A word $w \in \Sigma^*$ is *accepted* if there is an accepting path in the computation tree, though many non-accepting paths may exist. A NTM M accepts a word of length n in *time* $T(n)$ [resp. *space* $S(n)$] if the depth of the associated computation tree is at most $T(n)$ [resp. for each configuration α in the computation tree the number of cells used on each work tape is at most $S(n)$]. NTIME$(T(n))$ [resp. NSPACE$(S(n))$] is the collection of languages $L \subseteq \Sigma^*$ accepted by a NTM in time $O(T(n))$ [resp. space $O(S(n))$]; NP = NTIME$(n^{O(1)})$.

Similarly, NTimeSpace$(T(n), S(n))$ is the set of languages $L \subseteq \Sigma^*$ accepted by a NTM in time $O(T(n))$ and space $O(S(n))$.

With the previous definitions, any computation depending on all bits of the input requires at least linear time, the minimum amount of time taken to scan the input. However, by allowing a TM to access its input bitwise via pointers or random access, sublinear runtimes can be achieved, as shown by Chandra et al. [24].

Definition 2.3 A Turing machine M with *random access* (RATM) is given by a finite set Q of states, an input tape having no tape head, k work tapes, an *index query* tape and an *index answer* tape . To permit random access, the alphabet Γ is always assumed to contain the symbols $0, 1$. Except for the input tape, all other tapes have a tape head. M contains a distinguished input query state q_I, in which state M writes into the leftmost cell of the index answer tape that symbol which appears in the k-th input tape cell, where $k = \sum_{i<m} k_i \cdot 2^i$ is the integer whose binary representation is given by the contents

B	k_{m-1}	k_{m-2}	$\cdots$	k_0	B	$\cdots$

of the query index tape. Unlike the oracle Turing machine in Definition 2.5, the query index tape is not automatically erased after making an input bit query. A logtime RATM runs in time $O(\log n)$, where n is the length of the input.[3]

Logtime on a RATM is not so weak, and can compute certain simple functions, as shown by the next result. In the following, the function value $f(u) = v$ is computed by a logtime Turing machine in the sense that on input (k, u), the machine outputs the k-th bit of v in time logarithmic in the length of the input.

Fact 2.4 (Barrington-Immerman-Straubing [6]) Given an input of length n, a deterministic logtime RATM can

(i) compute the length of its input,
(ii) add and subtract integers of $O(\log n)$ bits,
(iii) decode a simple pairing function on strings of length $O(n)$.

Proof. Since a RATM has no output tape, we adopt the convention that M computes the function $f : \Sigma^* \to \Sigma^*$ if $|f(x)|$ is bounded by a polynomial in $|x|$, and for all bits, the i-th bit of $f(x)$ is a iff M accepts (x, i, a). The proof of *(i)* uses binary search, and according to [20], appears to have been first noticed by M. Dowd. The proof of *(ii)* is clear, since addition and subtraction take time linear in the input length. In *(iii)*, for $u, v \in \Sigma^*$ the pair (u, v) can be encoded by $\tau(|u|)11\tau(|v|)11uv$, where τ replaces each 0 [resp. 1] by 00 [resp. 01]. Decoding can then be done by using addition and random access. ■

A.M. Turing [137] introduced the notion of *relative computation* using an *oracle Turing machine* .

Definition 2.5 Let $B \subseteq \Gamma^*$. An oracle Turing machine (OTM) with oracle B is a Turing machine M which in addition to a read-only input tape, a distinguished output tape and finitely many work tapes, has a one-way infinite *oracle query tape* . The machine M has oracle answer states q_{yes} , q_{no} as well as a special oracle query state $q_?$ in which it queries whether the current contents of the oracle query tape belongs to oracle B. The transition function δ of M is a mapping from

$$(Q - \{q_A, q_R, q_?\}) \times (\Sigma \cup \{B\}) \times (\Gamma \cup \{B\})^{k+1}$$

into

$$Q \times (\Gamma \cup \{B\})^{k+1} \times \{-1, 0, 1\}^{k+2}.$$

A computation is defined as previously, except that if M is in state $q_?$ then the machine queries whether the word given by the current contents of the oracle query tape belongs to B. Dependent on the outcome of the oracle query, M goes into state q_{yes} or q_{no}, and simultaneously erases the query tape and places the oracle tape head at the leftmost square. This entire sequence of events takes place in one step. Finally, nondeterministic oracle Turing machines are analogously defined by adding the oracle apparatus to the NTM model.

For $A \subseteq \Sigma^*$ and $B \subseteq \Gamma^*$, write $A \leq_T B$ if A can be decided by an oracle Turing machine with oracle B. Similarly write $A \leq_T^P B$ [resp. $A \leq_T^{NP} B$] if A can be computed by a deterministic [resp. nondeterministic] oracle Turing machine with oracle B in polynomial time. Let $\Sigma_0^P = P$ and Σ_{n+1}^P be

$$\{A : (\exists B \in \Sigma_n^P)(A \leq_T^{NP} B)\}.$$

[3] Logarithms are with respect to base 2.

In [24], A. Chandra, D. Kozen and L. Stockmeyer introduced the *alternating Turing machine* (ATM) , a model suitable for formalizing divide and conquer algorithms.[4] When used with random access, this model allows sublinear runtimes and can be viewed as a kind of parallel computational device; in particular, uniform boolean circuit families, another parallel computation model, can be related to ATM's.

Definition 2.6 An alternating multitape Turing machine (ATM) M is specified by $(Q, \Sigma, \Gamma, \Delta, q_0, k, \ell)$ where $\ell : (Q - \{q_A, q_R\}) \to \{\wedge, \vee\}$ and $Q, \Sigma, \Gamma, \Delta, q_0, k$ are as in Definition 2.2 of a nondeterministic machine.

The function ℓ labels non-halting states as *universal* $(\wedge)$ and *existential* $(\vee)$. An *accepting computation tree* T is a subtree of the computation tree of M on x such that for any configuration $\alpha \in T$,

- the root of T is the initial configuration of M on x,
- if α is a leaf of T, then α is an *accepting* configuration,
- if α is universal, then for all β, $\alpha \vdash_M \beta \Rightarrow \beta \in T$, and
- if α is existential, then there exists $\beta \in T$ for which $\alpha \vdash_M \beta$.

The ATM M *accepts* input x if there is a non-empty accepting computation tree of M on x; otherwise x is rejected . $L(M)$ denotes the set of $x \in \Sigma^*$ accepted by M. The language $L(M)$ is accepted by M in *time* $T(n)$ [resp. *space* $S(n)$] if for each $w \in L(M)$ of length n, there is an accepting computation tree T of depth at most $T(n)$ [resp. in which at most $S(n)$ cells are used for each of the work tapes and index tapes at any node in the tree T]. The number of *alternations* M makes in an accepting computation tree T is defined to be the maximum number of alternations between existential and universal nodes in a path from the root to a leaf.

Convention 2.7 From now on, unless otherwise indicated, for any sublinear runtime $T(n) = o(n)$, the intended Turing machine model is RATM, while for runtimes $T(n) = \Omega(n)$, the intended Turing machine model is the conventional TM. This convention applies to deterministic, nondeterministic, and alternating Turing machines. While it is a simple exercise to show that PTIME is the same class, regardless of model, it appears to be an open problem to determine the relationship between DTIME$(T(n))$ on TM and RATM, for $T(n) = \Omega(n)$.

Definition 2.8

$$\begin{aligned}
\text{ATIME}(T(n)) &= \{L \subseteq \Sigma^* : L \text{ accepted by an ATM in time } O(T(n))\} \\
\text{ASPACE}(S(n)) &= \{L \subseteq \Sigma^* : L \text{ accepted by an ATM in space } O(S(n))\} \\
\text{ALOGTIME} &= \text{ATIME}(O(\log n)) \\
\text{APOLYLOGTIME} &= \cup_{k \geq 1} \text{ATIME}(O(\log^k n)) \\
\text{ALINTIME} &= \text{ATIME}(O(n)).
\end{aligned}$$

The *logtime hierarchy* LH [resp. the *linear time hierarchy* LTH , resp. the *polynomial time hierarchy* PH] is the collection of languages $L \subseteq \Sigma^*$, for which L is accepted by an ATM in time $O(\log n)$ [resp. $O(n)$, resp. $n^{O(1)}$] with at most $O(1)$ alternations.[5] Σ_k-TIME$(T(n))$ is the collection of languages accepted by an ATM in time $O(T(n))$ with at most k alternations, beginning with an existential state.

[4] Divide and conquer algorithms are generally space efficient. The *parallel computation thesis* states that sequential space equals parallel time (see [17]). In this sense, ATM's provide a parallel computation model.

[5] It follows from [51, 2] that LH is a hierarchy, where the collection of languages accepted by k-alternations is properly contained in the collection of languages accepted by $k+1$-alternations. The question of whether LTH or PH is a proper hierarchy is still open.

The class ALOGTIME is surprisingly powerful. It is not difficult to see that it contains all of the regular languages. To see this, recall that a finite state automaton M is a 5-tuple $(Q, q_0, \Sigma, \delta, F)$, where Q is a finite set of states, q_0 is the initial state, Σ a finite alphabet, $\delta : Q \times \Sigma \to Q$ and $F \subseteq Q$. For the empty word λ, let $\delta^*(\lambda) = q_0$, and for $w_1 \cdots w_n \in \Sigma^*$, let $\delta^*(w_1 \cdots w_n) = \delta(\delta^*(w_1 \cdots w_{n-1}), w_n)$. A word $w \in \Sigma^*$ is *accepted* by the finite state automaton M if $\delta^*(w) \in F$. The language L is *accepted* by M, denoted by $L = L(M)$, if it consists of the words accepted by M. Finally, a language L is *regular* if it is accepted by a finite state automaton.

Fact 2.9 If $L \subseteq \Sigma^*$ is regular, then $L \in$ ALOGTIME.

Proof. Suppose that L is recognized by a finite state automaton M given by $(Q, q_0, \Sigma, \delta, F)$. For each word w of Σ^*, associate the mapping $f_w : Q \to Q$ obtained by repeatedly applying the transition function δ on the letters from w. Formally, if $w = w_1 \cdots w_n$ then

$$f_w(q) = \delta(\cdots \delta(\delta(q, w_1), w_2), \ldots, w_n) \cdots).$$

When M is the minimal finite state automaton recognizing the regular language L, then the (finite) collection $\{f_w : w \in \Sigma^*\}$, constructed as above from M, is called the *syntactic monoid* of L.

Now, given the word $w = w_1 \cdots w_n \in \Sigma^*$, associate $f_{w_n}, \ldots, f_{w_1}$ with the leaves of a binary tree T, and at each internal node of T, compute the composition of two children nodes (the tree's root is at the top). The root of T then contains $f_w = f_{w_n} \circ \cdots \circ f_{w_1}$. It follows that $w \in L$ if and only if $f_w(q_0) \in F$. This construction can be formalized to yield an ALOGTIME algorithm. ∎

Much more striking are the results of D. Barrington[6] and S. Buss. First, define a language L to be *complete* for ALOGTIME under DLOGTIME reductions, if $L \in$ ALOGTIME and for any $L' \in$ ALOGTIME, there is a logtime many-one function f with the property that $|f(u)|$ is polynomial in $|u|$, and $u \in L'$ iff $f(u) \in L$. Here, the function value $f(u) = v$ is computed by a logtime Turing machine in the sense that on input (k, u), the machine outputs the k-th bit of v in time logarithmic in the length of the input.

Theorem 2.10 (D. Barrington [7]) *Let G be any finite non-solvable permutation group (for example S_5). Then the word problem*

$$\{(\sigma_1, \ldots, \sigma_n) : \sigma_i \in G, \sigma_1 \circ \sigma_2 \circ \cdots \circ \sigma_n = id\}$$

for G is complete for ALOGTIME *under* DLOGTIME *reductions.*

Sketch of Proof. If G is a group, then the *commutator* of elements $a, b \in G$ is the element $aba^{-1}b^{-1}$. The *commutator subgroup* of G, denoted by $[G, G]$, is the subgroup of G generated by all the commutators of G. For any group G, define $G^{(0)} = G$, and $G^{(n+1)} = [G^n, G^n]$. By definition, a group G is solvable if there is a finite series $G = G^{(0)} \geq G^{(1)} \geq \cdots \geq G^{(n)} = \{e\}$. If G is finite, then G is non-solvable if and only if $G = G^{(0)} \geq G^{(1)} \geq \cdots \geq G^{(n)} = G^{(n+1)} \neq \{e\}$, i.e. $G^{(n)}$ is non-trivial and equal to its commutator subgroup. For example, the groups A_k, S_k for $k \geq 5$ are non-solvable.

Assume now that G is a non-solvable group with series $G = G^{(0)} \geq \cdots \geq G^{(n)} = H$, and that H is non-trivial and equal to its commutator subgroup $[H, H]$; i.e. there exists m such that every element of H can be expressed as a product

[6] D. Barrington changed his name to D. Mix Barrington, so that some articles appear under the former name and some under the latter name.

$\Pi_{i=1}^m a_i b_i a_i^{-1} b_i^{-1}$ of m commutators of H. Using this observation, Barrington showed how to represent conjunctions and disjunctions as a word problem over H.

Namely, given a non-identity element $g \in H$ and an alternating AND/OR computation tree $T(x)$ for the computation of M on x, describe a word $w_{T(x)}$ in the elements of H such that

$$w_{T(x)} = \begin{cases} e & \text{if } M \text{ accepts } x \\ g & \text{else.} \end{cases}$$

This is done by induction on depth of node A in $T(x)$. Recall that $H = [H, H]$ and every element of H can be written as the product of m commutators of H. Then Barrington observed that if $A = (B \vee C)$ then

$$\begin{aligned} w_A(g) &= w_{B \vee C}(g) \\ &= \Pi_{i=1}^m w_B(b_i) w_C(c_i) w_B(b_i^{-1}) w_C(c_i^{-1}). \end{aligned}$$

Similarly, $B \wedge C$ and $\neg B$ can be expressed. Inductively one forms the word $w_{T(x)}$ whose product equals e exactly when M accepts x.

From the above discussion, with a close look at uniformity issues, it follows that the word problem for a finite non-solvable permutation group is hard for ALOGTIME. On the other hand, the word problem is clearly in ALOGTIME, since one can compose n permutations by associating them with the leaves of a binary tree, whose internal nodes compute the composition of their children. ∎

Theorem 2.11 (S. Buss [20, 22]) *The boolean formula valuation problem*

$$\{\Theta : \Theta \textit{ is a true variable-free propositional logic formula}\}$$

is complete for ALOGTIME *under* DLOGTIME *reductions.*

The proof of Theorem 2.11 is long and difficult, so cannot be sketched here. The results of Barrington and Buss are complementary in the sense that the word problem for S_5 is clearly in ALOGTIME, but not obviously complete, while the boolean formula evaluation problem is clearly complete but not obviously in ALOGTIME.

In [120], W. Savitch proved that $\text{NSPACE}(S(n)) \subseteq \text{DSPACE}(S^2(n))$, for any space constructible $S(n) \geq \log n$. The following theorem, due to Chandra, Kozen and Stockmeyer [24], is in part a generalization of Savitch's result that PSPACE = NPSPACE, and relates alternating time and space to deterministic time and space.

Theorem 2.12 (Chandra, Kozen, Stockmeyer [24]) *For* $f(n) \geq n$,

$$\text{ATIME}(f(n)) \subseteq \text{DSPACE}(f(n)) \subseteq \text{NSPACE}(f(n)) \subseteq \cup_{c>0}\text{ATIME}(c \cdot f(n)^2).$$

For $f(n) \geq \log n$,

$$\text{ASPACE}(f(n)) \subseteq \cup_{c>0}\text{DTIME}(c^{f(n)}).$$

From definitions, it is clear that

$$\text{LH} \subseteq \text{ALOGTIME} \subseteq \text{LOGSPACE} \subseteq \text{PTIME} \subseteq \text{PH} \subseteq \text{PSPACE}$$

and

$$\text{LH} \subseteq \text{LTH} \subseteq \text{ALINTIME} \subseteq \text{DLINSPACE} \subseteq \text{PSPACE}.$$

By Furst, Saxe, Sipser [51] and Ajtai [2], integer multiplication does not belong to LH (since multiplication $\times$ is a function, what is meant is that $\times \notin$ FLH, where the latter is the class of functions of polynomial growth rate, whose bitgraph belongs to LH; this is defined later). Note that Buss [21] has even shown that the graph

of multiplication does not belong to LH. Since the graph of integer multiplication belongs to ALOGTIME, the first containment above is proper. With this exception, nothing else is known about whether the other containments are proper.

All the previous machine models concern language recognition problems. Predicates $R \subseteq (\Sigma^*)^k$ can be recognized by allowing input of the form

$$Bx_1Bx_2B\cdots Bx_nB$$

consisting of n inputs $x_i \in \Sigma^*$, each separated by the blank $B \notin \Sigma$. By adding a write-only output tape with a tape head capable only of writing and moving to the right, and by allowing input of the form $Bx_1Bx_2B\cdots Bx_nB$, a TM or RATM can compute an n-place function. In the literature, function classes such as the polynomial time computable functions were so introduced. To provide uniform notation for such function classes, along with newer classes of sublinear time computable functions, we proceed differently.

Definition 2.13 A function $f(x_1,\ldots,x_n)$ has *polynomial growth* resp. *linear growth* resp. *logarithmic growth* if

$$|f(x_1,\ldots,x_n)| = O(\max_{1\leq i\leq n}|x_i|^k), \text{for some } k$$

resp.

$$|f(x_1,\ldots,x_n)| = O(\max_{1\leq i\leq n}|x_i|)$$

resp.

$$|f(x_1,\ldots,x_n)| = O(\log(\max_{1\leq i\leq n}|x_i|).$$

The *graph* G_f satisfies $G_f(\vec{x},y)$ iff $f(\vec{x}) = y$. The *bitgraph* B_f satisfies $B_f(\vec{x},i)$ iff the i-th bit of $f(\vec{x})$ is 1. If $\mathcal{C}$ is a complexity class, then $\mathcal{FC}$ [resp. $Lin\mathcal{FC}$ resp. $Log\mathcal{FC}$] is the class of functions of polynomial [resp. linear resp. logarithmic] growth whose bitgraph belongs to $\mathcal{C}$. In this paper, $\mathcal{GC}$ will abbreviate $Lin\mathcal{FC}$. The iteration $f^{(n)}(x)$ is defined by induction on n: $f^{(0)}(x) = x$, $f^{(n+1)}(x) = f(f^{(n)}(x))$. With this notation, the iteration $\log^{(n)} x$ should not be confused with the power $\log^n x = (\log x)^n$.

There are other extensions of the Turing machine model not covered in this survey, such as the *probabilistic* Turing machine (yielding classes such as R and BPP, see [139]), the *genetic* Turing machine (defined by P. Pudlák [113], who showed that polynomial time bounded genetic TM's compute exactly PSPACE), and the *quantum* Turing machine (first introduced by D. Deutsch [47], and for which P. Shor [129] proved that integer factorization is computable in bounded error probabilistic quantum polynomial time BQP).

2.2 Parallel machine model

> "Having one processor per data element changes the way one thinks."
> W.D. Hillis and G.L. Steele, Jr. [71]

Emerging around 1976-77 from the work of Goldschlager [56, 57], Fortune-Wyllie [50], and Shiloach-Vishkin [128], the *parallel random access machine* (PRAM) provides an abstract model of parallel computation for algorithm development. While existent "massively parallel" computers generally require a specific communication network (e.g. hypercube, mesh, etc.) for message passing between processors (and such details are of immense practical importance), the PRAM abstracts out all such processor communication details and postulates a global shared memory . Individual processors of a PRAM additionally have local memory, and while operating

synchronously on the same program, are capable of performing arithmetic and logical operations as well as local and global read/write in both direct and indirect addressing mode. Processors may have different data stored in their local memories and have access to their unique processor identity number PID . Thus the effect of an instruction like "add the contents of the PID-th global memory register to local memory register 2 and store in local memory register 7" may be quite different in different processors. Different models of PRAM have been studied, depending on the strength of local arithmetic operations allowed, and whether simultaneous read/write in the same global memory register is allowed by several processors. This yields EREW, CREW, and CRCW models, according to whether exclusive read, exclusive write, concurrent read or concurrent write are allowed. An excellent survey of parallel algorithms and models is R.M. Karp and V. Ramachandran [81]. The formal development follows.

A *concurrent random access machine* CRAM has a sequence $R_0, R_1, \ldots$ of random access machines which operate in a synchronous fashion in parallel. Each R_i has its own local memory, an infinite collection of registers, each of which can hold an arbitrary non-negative integer. Global memory consists of an infinite collection of registers accessible to all processors, which are used for reading the input, processor message passing, and output. Global registers are designated $M_0^g, M_1^g, M_2^g, \ldots$, and local registers by $M_0, M_1, M_2, \ldots$ – local registers of processor P_i might be denoted $M_{i,0}, M_{i,1}, \ldots$. A global memory register can be read simultaneously by several processors (*concurrent read*, rather than *exclusive read*). In the case where more than one processor may attempt to write to the same global memory register, the lowest numbered processor succeeds (*priority resolution* of write conflict in this *concurrent write* rather than *exclusive write* model). An input x is initially given bitwise in the global registers, the register M_i^g holding the i-th bit of x. All other registers initially contain the blank symbol B (different from $0, 1$) which designates that the register is empty. Similarly at termination, the output y is specified in the global memory, the register M_i^g holding the i-th bit of y. At termination of a computation all other global registers contain the blank symbol. [The input/output convention of one integer per global memory register yields an equivalent model for the complexity classes here considered.] Let *res* (result), *op*0, *op*1, *op*2 (operands 0,1,2) be non-negative integers. If any register occurring on the right side of an instruction contains 'B', then the register on the left side of the instruction will be assigned the value 'B' (undefined).

Instructions are as follows.

M_{res} = `constant`

M_{res} = `processor number`

M_{res} = M_{op1}

M_{res} = $M_{op1} + M_{op2}$

M_{res} = $M_{op1} \dot{-} M_{op2}$

M_{res} = `MSP`(M_{op1}, M_{op2})

M_{res} = `LSP`(M_{op1}, M_{op2})

M_{res} = $*M_{op1}$

M_{res} = $*M_{op1}^g$

$*M_{res}$ = M_{op1}

$*M_{res}^g$ = M_{op1}

`GOTO label`

`GOTO label IF` M_{op1} = M_{op2}

`GOTO label IF` $M_{op1} \leq M_{op2}$

`HALT`

Cutoff subtraction is defined by $x \dot{-} y = x - y$, provided that $x \geq y$, else 0. The shift operators MSP and LSP are defined by

- $\texttt{MSP}(x, y) = \lfloor x/2^y \rfloor$, provided that $y < |x|$, otherwise 'B',
- $\texttt{LSP}(x, y) = x - 2^y \cdot (\lfloor x/2^y \rfloor)$, provided that $y \leq |x|$, otherwise 'B'.

The CRAM model is due to N. Immerman [74], though there slightly different conventions are made.

Instructions with '$*$' concern indirect addressing. The instruction $M_{res} = *M_{op1}$ assigns to local register M_{res} the contents of local register with address given by the value M_{op1}. Similarly, $M_{res} = *M^g_{op1}$ performs an indirect read from global memory into local memory. The instruction $*M_{res} = M_{op1}$ assigns the value of local register M_{op1} to the local register whose address is given by the current contents of the local register M_{res}. Similarly, $*M^g_{res} = M_{op1}$ performs an indirect write into global memory.

In summary, the CRAM has instructions for *(i)* local operations — addition, cutoff subtraction, shift, *(ii)* global and local indirect reading and writing, *(iii)* control instructions — GOTO, conditional GOTO and HALT. A program is a finite sequence of instructions, where each individual processor of a CRAM has the same program. Each instruction has unit-cost (*uniform time cost*). During the course of a computation, only finitely many *active* processors perform computations. An input x of length n is *accepted* by a CRAM M in time $T(n)$ with $P(n)$ many active processors, if M halts after $T(n)$ time where processors $P_0, \ldots, P_{n-1}$ synchronously execute the program. The class TIMEPROC$(T(n), P(n))$ consists of those languages accepted by a CRAM in time $T(n)$ with $P(n)$ many processors.

Example 2.14 The following is a CRAM program for computing $|x| = \lceil \log_2(x+1) \rceil$, where comments begin by '%'.

Let $M_{res} = \text{BIT}(M_{op1}, M_{op2})$ be the instruction which, for $i = M_{op2}$ computes the coefficient of 2^i in the binary representation of the integer stored in M_{op1}, provided that $i < |M_{op1}|$, and otherwise returns the value 'B'.

```
1  M1 = processor number
2  M2 = *M1^g    % in Pi, Mi = Mi^g
3  if (M2 = B) then M0^g = M1
4  M3 = M0^g     % in Pi, M3 = least i [ Mi^g = B ] = |x|
5  *M1^g = B      % erase global memory
6  M4 = 1
7  M4 = M1 + 1
8  M4 = M3 ∸ M4
9  M5 = MSP(M3,M4)
10 M6 = MSP(M5,1)
11 M6 = M6 + M6
12 M4 = M5 ∸ M6
13  *M1^g = M4  % output placed in global memory
14  HALT
```

Processor bound: $P(|x|) = |x|$.

Strictly speaking, line 3 is not syntactically allowed, but can easily be implemented with a few extra lines of code, and will not affect the time or processor bound.

Lines 6–12 ensure that $M_4 = \mathrm{BIT}(M_3, M_3 \dot{-} (M_1 + 1))$, so that in processor P_i, $M_4 = \mathrm{BIT}(|x|, |x| \dot{-} (i + 1))$.

To further illustrate the CRAM model, Algorithm 2.15 computes $\max(x_1, \ldots, x_n)$ of n integers in constant time with $O(n^2)$ processors.

Algorithm 2.15 Constant time algorithm for maximum.

(1) for all $\binom{n}{2}$ pairs $1 \leq i < j \leq n$ in parallel do

$$a_{i,j} = \begin{cases} 1 & \text{if } x_i < x_j \\ 0 & \text{else} \end{cases}$$

(2) for $i := 1$ to n in parallel do
  $m_i := 0$
(3) for $1 \leq i < j \leq n$ in parallel do
  if $a_{i,j} = 1$ then $m_i := 1$
(4) for $i := 1$ to n in parallel do
  if $m_i = 0$ then $m := i$
(5) $\max := x_m$

Time $= O(1)$, **Processors** $= O(n^2)$

2.3 Circuit families

A directed graph G is given by a set $V = \{1, \ldots, m\}$ of vertices (or nodes) and a set $E \subseteq V \times V$ of edges. The *in-degree* or *fan-in* [resp. *out-degree* or *fan-out*] of node x is the size of $\{i \in V : (i, x) \in E\}$ [resp. $\{i \in V : (x, i) \in E\}$]. A *circuit* C_n is a labeled, directed acyclic graph whose nodes of in-degree 0 are called *input* nodes and are labeled by one of $0, 1, x_1, \ldots, x_n$, and whose nodes v of in-degree $k > 0$ are called *gates* and are labeled by a k-place function from a *basis* set of boolean functions. A circuit has a unique *output* node of out-degree 0.[7] A family $\mathcal{C} = \{C_n : n \in \mathbf{N}\}$ of circuits has *bounded fan-in* if there exists k, for which all gates of all C_n have in-degree at most k; otherwise $\mathcal{C}$ has *unbounded* or *arbitrary* fan-in.

Boolean circuits have basis $\wedge, \vee, \neg$, where $\wedge, \vee$ may have fan-in larger than 2 (as described below, the AC^k [resp. NC^k] model concerns unbounded fan-in [resp. fan-in 2] boolean circuits). A *threshold* gate $\mathrm{TH}_{k,n}$ outputs 1 if at least k of its n inputs is 1. A *modular counting* gate $\mathrm{MOD}_{k,n}$ outputs 1 if the sum of its n inputs is evenly divisible by k. A *parity* gate $\oplus$ outputs 1 if the number of input bits equal to 1 is even, where as for $\wedge, \vee$ the fan-in may be restricted to 2, or arbitrary, depending on context.

An input node v labeled by x_i computes the boolean function $f_v(x_1, \ldots, x_n) = x_i$. A node v having in-edges from $v_1, \ldots, v_m$, and labeled by the m-place function g from the basis set, computes the boolean function $f_v(x_1, \ldots, x_n) = g(f_{v_1}(x_1, \ldots, x_n), \ldots, f_{v_m}(x_1, \ldots, x_n))$. A circuit C_n *accepts* the word $x_1 \cdots x_n \in \{0,1\}^n$ if $f_v(x_1, \ldots, x_n) = 1$, where f_v is the function computed by the unique output node v of C_n. A family $(C_n : n \in \mathbf{N})$ of circuits *accepts* a language $L \subseteq \{0,1\}^*$ if for each n, $L^n = L \cap \{0,1\}^n$ consists of the words accepted by C_n.

The *depth* of a circuit is the length of the longest path from an input to an output node, while the *size* is the number of gates. A language $L \subseteq \{0,1\}^*$ belongs to $\mathrm{SIZEDEPTH}(S(n), D(n))$ over basis B if L consists of those words accepted by a family $(C_n : n \in \mathbf{N})$ of circuits over basis B, where $size(C_n) = O(S(n))$ and $depth(C_n) = O(D(n))$.

[7]The usual convention is that a circuit may have any number of output nodes, and hence compute a function $f : \{0,1\}^n \to \{0,1\}^m$. In this paper, we adopt the convention that a circuit computes a boolean function $f : \{0,1\}^n \to \{0,1\}$. An m-output circuit C computing function $g : \{0,1\}^n \to \{0,1\}^m$ can then be simulated by a circuit computing the boolean function $f : \{0,1\}^{n+m} \to \{0,1\}$ where $f(x_1, \ldots, x_n, 0^{m-i}1^i) = 1$ iff the i-th bit of $g(x_1, \ldots, x_n)$ is 1.

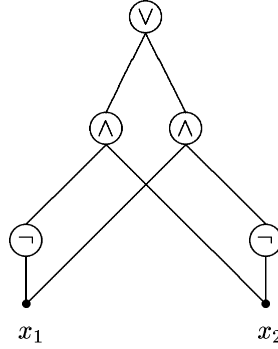


Figure 1: Exclusive-or

A boolean circuit which computes the function $f(x_1, x_2) = x_1 \oplus x_2$ is as in Figure 1.

Example 2.16 The function $\max(a_0, \ldots, a_{n-1})$ of n integers, each of size at most m, can be computed by a boolean circuit as follows. Assume the integers a_i are distinct (a small modification is required for non-distinct integers). Then the k-th bit of $\max(a_0, \ldots, a_{n-1})$ is 1 exactly when

$$(\exists i < n)(\forall j < n)(j \neq i \rightarrow a_j \leq a_i \wedge \text{BIT}(k, a_i) = 1).$$

This bounded quantifier formula is translated into a boolean circuit by

$$\bigvee_{i<n} \bigwedge_{j<n, j\neq i} \bigvee_{\ell<n} \bigwedge_{\ell<p<m} (\text{BIT}(p, a_j) = \text{BIT}(p, a_i) \wedge \text{BIT}(\ell, a_j) = 0 \wedge \text{BIT}(\ell, a_i) = 1.$$

Note that by Algorithm 2.15, $\max(a_0, \ldots, a_{n-1})$ is computed by a CRAM in constant time with a polynomial number of processors and by Example 2.16 $\max(a_0, \ldots, a_{n-1})$ is computed by a constant depth polynomial size family of boolean circuits. As this suggests, there is a relation between time/processors for a CRAM and depth/size for a boolean circuit family. The exact relation between the two models is given in Theorem 2.18.

Without a uniformity condition, circuit families of depth 2 and size 1 can accept non-recursive languages (e.g. all inputs are accepted [resp. rejected] if the n-th circuit is of the form $x_1 \vee \neg x_1$ [resp. $x_1 \wedge \neg x_1$]). Various notions of uniformity have been suggested (PTIME-uniformity [8], LOGSPACE-uniformity [17], U_{E^*}-uniformity [119], etc.), but the most robust (and strictest) appears to be that of LOGTIME-uniformity [18, 6], which is adopted in this paper.

Definition 2.17 (W. Ruzzo [119], also [6]) The *direct connection language* (DCL) of a circuit family $(C_n : n \in \mathbf{N})$ is the set of $(a, b, \ell, 0^n)$, where a is the parent of b in the circuit C_n, and the label of gate a is ℓ. A circuit family is LOGTIME-uniform if its associated DCL belongs to DLOGTIME. For $k \geq 0$, AC^k [resp. NC^k] is the class of languages accepted by LOGTIME-uniform $\text{SIZEDEPTH}(n^{O(1)}, O(\log^k n))$ over the boolean basis, where $\wedge$,$\vee$ have arbitrary fan-in [resp. fan-in 2], and $\text{NC} = \cup_k \text{AC}^k = \cup_k \text{NC}^k$. $\text{ACC}(k)$ is the class of languages accepted by LOGTIME-uniform $\text{SIZEDEPTH}(n^{O(1)}, O(1))$ over the basis $\wedge$, $\vee$, $\neg$, $\text{MOD}_{k,n}$, where $\wedge$,$\vee$ have unbounded

fan-in, and $\text{ACC} = \cup_{k\geq 2}\text{ACC}(k)$. TC^0 is the class of languages in LOGTIME-uniform $\text{SIZEDEPTH}(n^{O(1)}, O(1))$ over the basis $\text{TH}_{k,n}$.[8]

In [133], L. Stockmeyer and U. Vishkin related PRAM time and processors to boolean circuit depth and size. The LOGTIME-uniform version of that result was proved by N. Immerman [74] and follows.

Theorem 2.18 *For $k \geq 0$, AC^k equals $\text{TIMEPROC}(O(\log^k n), n^{O(1)})$ on a CRAM.*

The following containments are known:

$$\text{NC}^k \subseteq \text{AC}^k \subseteq \text{NC}^{k+1}$$

and

$$\text{NC}^1 = \text{ALOGTIME} \subseteq \text{LOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{AC}^1.$$

None of the inclusions are known to be strict or not. For more information on circuits, see the excellent survey article by R. Boppana and M. Sipser [16].

From this point on, we will assume that all language and function complexity classes are over the alphabet $\{0, 1\}$.

3 Some recursion schemes

Kleene's normal form theorem [82] states that for each recursive (partial) function f there is an index e for which $f(\vec{x}) = U(\mu y[T(e, \vec{x}, y) = 0])$, where T, U are primitive recursive. The proof relies on arithmetizing computations via Gödel numbers, a technique introduced in [53] by Gödel, and with which Turing computable functions can be shown equivalent to μ-recursive functions. Since then, there have been a number of *arithmetizations* of machine models [82, 83, 13, 40, 114, 135, 141, 39, 37], etc. Key to all of these results is the availability in a function algebra $\mathcal{F}$ of a conditional function, a pairing function, and some string manipulating functions, in order to show that the function $\text{NEXT}_M(x, c) = d$ belongs to $\mathcal{F}$. Here, c, d encode configurations of machine M on input x and d is the configuration obtained in one step from configuration c.

Definition 3.1 An *operator* (here also called *operation*) is a mapping from functions to functions. If $\mathcal{X}$ is a set of functions and OP is a collection of operators, then $[\mathcal{X}; \text{OP}]$ denotes the smallest set of functions containing $\mathcal{X}$ and closed under the operations of OP. The set $[\mathcal{X}; \text{OP}]$ is called a *function algebra*. In a straightforward inductive manner, define *representations* or *names* for functions in $[\mathcal{X}; \text{OP}]$. The *characteristic function* $c_P(\vec{x})$ of a predicate P satisfies

$$c_p(\vec{x}) = \begin{cases} 1 & \text{if } P(\vec{x}) \\ 0 & \text{else,} \end{cases} \tag{1}$$

where P is often written in place of c_P. If $\mathcal{F}$ is a class of functions, then $\mathcal{F}_*$ is the class of predicates whose characteristic function belongs to $\mathcal{F}$.[9]

[8] In this paper, circuit classes such as AC^k, NC^k, NC and TC^0 sometimes denote both language classes, though more often function classes, where the intended meaning is clear from context. That is, we write NC in place of $\mathcal{F}\text{NC}$, etc. NC is an acronym for "Nick's Class", as this class was first studied by N. Pippenger. AC^k was studied by W.L. Ruzzo, using the alternating Turing machine model.

[9] In [58], Grzegorczyk defined $\mathcal{F}_*$ as the collection of predicates P for which there is a function $f \in \mathcal{F}$ satisfying $P(\vec{x}) \iff f(\vec{x}) = 0$. For the function classes here considered, these are equivalent definitions.

Definition 3.2 Let $\mathcal{F} = [f_1, f_2, \ldots; O_1, O_2, \ldots]$ be a function algebra. Let O denote operator O_{i_0}, and fix a representation R of $f \in \mathcal{F}$. The rank $rk_{O,R}(f)$ of applications of O in the representation R of $f \in \mathcal{F}$ is defined by induction. If f is an initial function $f_1, f_2, \ldots$ then $rk_{O,R}(f) = 0$. Suppose that f is defined by application of operator O_i to functions $g_1, \ldots, g_m$ where $rk_{O,R}(g_j) = r_j$ for $1 \leq j \leq m$. If $i = i_0$ then $rk_{O,R}(f) = 1 + \max\{r_1, \ldots, r_m\}$; otherwise $rk_{O,R}(f) = \max\{r_1, \ldots, r_m\}$. The *$O$-rank* $rk_O(f)$ of a function $f \in \mathcal{F}$ is the minimum of $rk_{O,R}(f)$ over all representations R of f in $\mathcal{F}$.

Operations which have been studied in the literature include composition, primitive recursion, minimization, and their variants including bounded composition, bounded recursion, bounded recursion on notation, bounded minimization, simultaneous recursion, multiple recursion, course-of-values recursion, divide and conquer recursion, safe and tiered recursion, etc.[10] Good surveys of function algebras include the monographs by H. Rose [117] and K. Wagner and G. Wechsung [142] (chapters 2, 10).

Since newer results concerning smaller complexity classes yield older results concerning larger classes as corollaries, we begin with a function algebra introduced by the author for the class $\mathcal{F}$LH of functions in the logtime hierarchy. By [6], this class is equal to the class AC^0 of functions computable on a concurrent random access machine in constant parallel time with a polynomial number of processors.

3.1 An algebra for the logtime hierarchy LH

Definition 3.3 The *successor* function satisfies $s(x) = x + 1$; the *binary successor* functions s_0, s_1 satisfy $s_0(x) = 2 \cdot x$, $s_1(x) = 2 \cdot x + 1$; the n-place projection functions $I^n_k(x_1, \ldots, x_n) = x_k$; I denotes the collection of all projection functions.

Definition 3.4 The function f is defined by *composition* (COMP) from the functions $h, g_1, \ldots, g_m$ if

$$f(x_1, \ldots, x_n) = h(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)).$$

The function f is defined by *primitive recursion* (PR) from functions g, h if

$$\begin{aligned} f(0, \vec{y}) &= g(\vec{y}), \\ f(x+1, \vec{y}) &= h(x, \vec{y}, f(x, \vec{y})). \end{aligned}$$

The collection $\mathcal{PR}$ of primitive recursive functions is $[0, I, s; \text{COMP}, \text{PR}]$. The function f is defined by *iteration* (ITER) from functions g if

$$\begin{aligned} f(0) &= 0 \\ f(x+1) &= g(f(x)). \end{aligned}$$

Theorem 3.5 (R.M. Robinson [116]) *Define the operation* ADD *by* $\text{ADD}(f, g)(x) = f(x) + g(x)$, *and let* $q(x) = x - \lfloor \sqrt{x} \rfloor^2$. *Let* $\mathcal{PR}_1$ *denote the collection of one-place primitive recursive functions. Then* $\mathcal{PR}_1$ *equals* $[0, s, q; \text{COMP}, \text{ITER}, \text{ADD}]$.

In [4] G. Asser presented a version of the previous theorem for primitive recursive word functions of one variable.

Primitive recursion defines $f(x+1)$ in terms of $f(x)$, so that the computation of $f(x)$ requires approximately $2^{|x|}$ many steps, an exponential number in the length of x. To define smaller complexity classes of functions, Bennett [13] introduced the scheme of *recursion on notation*, which Cobham [40] later used to characterize the polynomial time computable functions.

[10] In this paper, for uniformity of notation, a number of operations are introduced as *bounded* instead of *limited* operations. For example, Grzegorczyk's schemes of *limited recursion* and *limited minimization* are here called *bounded recursion* and *bounded minimization*.

Definition 3.6 Assume that $h_0(x,\vec{y}), h_1(x,\vec{y}) \leq 1$. The function f is defined by *concatenation recursion on notation* (CRN) from g, h_0, h_1 if

$$\begin{aligned} f(0,\vec{y}) &= g(\vec{y}) \\ f(s_0(x),\vec{y}) &= s_{h_0(x,\vec{y})}(f(x,\vec{y})), \quad \text{if } x \neq 0 \\ f(s_1(x),\vec{y}) &= s_{h_1(x,\vec{y})}(f(x,\vec{y})). \end{aligned}$$

This scheme can be written in the abbreviated form

$$\begin{aligned} f(0,\vec{y}) &= g(\vec{y}) \\ f(s_i(x),\vec{y}) &= s_{h_i(x,\vec{y})}(f(x,\vec{y})). \end{aligned}$$

The scheme CRN was introduced by the author in [39], though motivated by a similar scheme due to J. Lind [94]. If concatenation of the empty string is allowed, or if the condition $h_i(x,\vec{y}) \leq 1$ is dropped (i.e. if concatenation of $f(x,\vec{y})$ with an arbitrary string $h_i(x,\vec{y})$ is allowed as in Lind's scheme), then the resulting scheme is provably stronger (i.e. parity $\oplus_{i=1}^n x_i$ is easily defined using Lind's version, although parity is known not to belong to LH).

Definition 3.7 The length of x in binary satisfies $|x| = \lceil \log(x+1) \rceil$; $||x||$ is defined as $|(|x|)|$, etc.; $\text{MOD}2(x) = x - 2 \cdot \lfloor \frac{x}{2} \rfloor$; the function $\text{BIT}(i,x) = \text{MOD}2(\lfloor \frac{x}{2^i} \rfloor)$ yields the coefficient of 2^i in the binary representation of x; the *smash* function satisfies $x\#y = 2^{|x|\cdot|y|}$. The algebra A_0 is defined to be

$$[0, I, s_0, s_1, \text{BIT}, |x|, \#; \text{COMP}, \text{CRN}].$$

Arbitrary constants belong to A_0. For instance the integer 6 has binary representation 110 and is represented by $s_0(s_1(s_1(0)))$. The auxiliary reverse function $rev0(x,y)$ gives the $|y|$ many least significant bits of x written in reverse. Let

$$\begin{aligned} rev0(x,0) &= 0 \\ rev0(x,s_i(y)) &= s_{\text{BIT}(|y|,x)}(rev0(x,y)). \end{aligned} \tag{2}$$

The reverse of the binary notation for x is given by $rev(x) = rev0(x,x)$. For instance the integer 10 has binary notation 1010 whose reverse is 101, corresponding to the integer 5, so $rev(10) = 5$. The following computation may be helpful, where $\overline{w}$ temporarily denotes the integer having binary representation w.

$$\begin{aligned} rev(10) &= rev(\overline{1010}) \\ &= rev0(\overline{1010},\overline{1010}) \\ &= s_{\text{BIT}(|\overline{101}|,\overline{1010})}(rev0(\overline{1010},\overline{101})) \\ &= s_1(s_{\text{BIT}(|\overline{10}|,\overline{1010})}(rev0(\overline{1010},\overline{10}))) \\ &= s_1 s_0(s_{\text{BIT}(|\overline{1}|,\overline{1010})}(rev0(\overline{1010},\overline{1}))) \\ &= s_1 s_0 s_1(s_{\text{BIT}(\overline{0},\overline{1010})}(rev0(\overline{1010},\overline{0}))) \\ &= s_1 s_0 s_1 s_0(0) \\ &= 5 \end{aligned}$$

Let

$$\begin{aligned} ones(0) &= 0 \\ ones(s_i(x)) &= s_1(ones(x)) \end{aligned} \tag{3}$$

so that $ones(x) = 2^{|x|} - 1$ whose binary representation consists of $|x|$ many 1's. Let

$$\begin{aligned} pad(x,0) &= x \\ pad(x,s_i(y)) &= s_0(pad(x,y)) \end{aligned} \tag{4}$$

so that $pad(x,y) = 2^{|y|} \cdot x$ whose binary representation is that of x with $|y|$ many 0's appended to the right. Kleene's signum functions sg, $\overline{sg}$, which satisfy $sg(x) = \min(x,1)$ and $\overline{sg}(x) = 1 - sg(x)$, are defined by

$$\begin{aligned} sg(x) &= \text{BIT}(0, ones(x)) \\ \overline{sg}(x) &= \text{BIT}(0, pad(1,x)). \end{aligned} \tag{5}$$

The conditional function, easily defined using stronger recursion schemes,

$$cond(x,y,z) = \begin{cases} y & \text{if } x = 0 \\ z & \text{else} \end{cases} \tag{6}$$

is here defined using the auxiliary functions $cond0$, $cond1$, $cond2$. Define

$$\begin{aligned} cond0(0,y) &= 0 \\ cond0(s_i(x),y) &= s_{\text{BIT}(0,y)}(cond0(x,y)) \end{aligned} \tag{7}$$

$$cond1(x,y) \quad = \quad sg(cond0(x,y)) \tag{8}$$

$$\begin{aligned} cond2(x,0) &= 0 \\ cond2(x,s_i(y)) &= s_{cond1(x,s_i(y))}(cond2(x,y)) \end{aligned} \tag{9}$$

so that

$$cond0(x,y) = \begin{cases} 0 & \text{if BIT}(0,y) = 0 \\ 2^{|x|} - 1 & \text{else} \end{cases} \tag{10}$$

$$cond1(x,y) = \begin{cases} 0 & \text{if } x = 0 \\ \text{BIT}(0,y) & \text{else} \end{cases} \tag{11}$$

$$cond2(x,y) = \begin{cases} 0 & \text{if } x = 0 \\ y & \text{else.} \end{cases} \tag{12}$$

The concatenation function $x * y = 2^{|y|} \cdot x + y$ is defined by

$$\begin{aligned} x * 0 &= x \\ x * s_i(y) &= s_i(x * y). \end{aligned} \tag{13}$$

Then the conditional function $cond$ is defined by

$$cond(x,y,z) = cond2(\overline{sg}(x), y) * cond2(x,z).$$

With $cond$ one can form (characteristic functions of) predicates by applying boolean operations AND, OR, NOT to other predicates. Additionally, using $cond$, one can introduce functions using *definition by cases*

$$f(\vec{x}) = \begin{cases} g_1(\vec{x}) & \text{if } P_1(\vec{x}) \\ g_2(\vec{x}) & \text{if } P_2(\vec{x}) \\ & \vdots \\ g_n(\vec{x}) & \text{if } P_n(\vec{x}) \end{cases} \tag{14}$$

where predicates $P_1, \ldots, P_n$ are disjoint and exhaustive. A *sharply bounded quantifier* is of the form $(\exists x \leq |y|)$ or $(\forall x \leq |y|)$.

Lemma 3.8 *$(A_0)_*$ is closed under sharply bounded quantifiers.*

Proof. Suppose that the predicate $R(x, \vec{z})$ belongs to A_0 and that $P(y, \vec{z})$ is defined by $(\exists x \leq |y|)R(x, \vec{z})$. Define

$$f(0, \vec{z}) = 0$$
$$f(s_i(x), \vec{z}) = s_{R(|x|, \vec{z})}(f(x, \vec{z})).$$

Then $P(y, \vec{z}) = sg(f(s_1(y), \vec{z}))$ belongs to A_0. Bounded universal quantification can be derived from bounded existential quantification using $\overline{sg}$. ■

Definition 3.9 The function f is defined by *sharply bounded minimization* (SBMIN) from the function g, denoted by $f(x, \vec{y}) = \mu i \leq |x|[g(i, \vec{y}) = 0]$, if

$$f(x, \vec{y}) = \begin{cases} \min\{i \leq |x| : g(i, \vec{y}) = 0\} & \text{if such exists} \\ 0 & \text{else.} \end{cases}$$

Sharply bounded maximization (SBMAX) is analogously defined.

From Lemma 3.8, it follows that A_0 is closed under SBMIN and SBMAX . Namely, define

$$k(0, \vec{y}) = 0$$
$$k(s_i(z), \vec{y}) = s_{h(z, \vec{y})}(k(z, \vec{y}))$$

where

$$h(z, \vec{y}) = \begin{cases} 0 & \text{if } (\exists x \leq |z|)[g(x, \vec{y}) = 0] \\ 1 & \text{else.} \end{cases}$$

Then $f(x, \vec{y}) = \mu i \leq |x|[g(i, \vec{y}) = 0]$ is defined by

$$f(x, \vec{y}) = \begin{cases} 0 & \text{if } g(0, \vec{y}) = 0 \text{ or } \neg(\exists i \leq |x|)[g(i, \vec{y}) = 0] \\ |rev(k(s_1(x), \vec{y}))| & \text{else.} \end{cases}$$

The integer x is a beginning of y, denoted xBy, if the binary representation of x is an initial segment (from left to right) of the binary representation of y; formally xBy iff $x = 0$ or $x, y > 0$ and

$$(\forall i \leq |x|)[\text{BIT}(i, rev(s_1(x))) = \text{BIT}(i, rev(s_1(y)))].$$

Thus the predicate $B \in A_0$. Similarly, predicates xPy (x is part of y, i.e. a convex subword of y) and xEy (x is an end of y) can be shown to belong to A_0.

To show the closure of A_0 under part-of quantifiers $(\exists xBy)$, $(\exists xPy)$, $(\exists xEy)$, etc. define the *most significant part* function MSP by

$$\text{MSP}(0, y) = 0$$
$$\text{MSP}(s_i(x), y) = s_{\text{BIT}(y, s_i(x))}(\text{MSP}(x, y)) \tag{15}$$

and the *least significant part* function LSP by

$$\text{LSP}(x, y) \quad = \quad \text{MSP}(rev(\text{MSP}(rev(s_1(x)), |\text{MSP}(x, y)|)), 1). \tag{16}$$

These functions satisfy $\text{MSP}(x, y) = \lfloor \frac{x}{2^y} \rfloor$ and $\text{LSP}(x, y) = x \bmod 2^y$, where $x \bmod 1$ is defined to be 0. For later reference, define the unary analogues msp, lsp by

$$msp(x, y) = \lfloor x/2^{|y|} \rfloor = \text{MSP}(x, |y|) \tag{17}$$
$$lsp(x, y) = x \bmod 2^{|y|} = \text{LSP}(x, |y|), \tag{18}$$

and note that lsp is definable from msp, rev as follows

$$lsp(x, y) \quad = \quad msp(rev(msp(rev(s_1(x)), msp(x, y))), 1). \tag{19}$$

Using MSP, LSP together with ideas of the proof of the previous lemma, the following is easily shown.

Lemma 3.10 *$(A_0)_*$ is closed under part-of quantifiers.*

Using part-of quantification, the inequality predicate $x \le y$ can be defined by

$$|x| < |y|$$

$$\text{OR}$$

$$|x| = |y| \text{AND} (\exists u B x)[uBy \wedge \text{BIT}(|x| \dot{-} |u| \dot{-} 1, y) = 1 \wedge \text{BIT}(|x| \dot{-} |u| \dot{-} 1, x) = 0]$$

where $|x| < |y|$ has characteristic function $sg(\text{MSP}(y, |x|))$. Note that $|x| \dot{-} |u| \dot{-} 1$ can be expressed by $|msp(msp(x,u),1)| = |\lfloor \frac{msp(x,u)}{2} \rfloor|$.

Addition $x + y$ can be defined in A_0 by applying CRN to $sum(x,y,z)$, whose value is the $|z|$-th bit of $x+y$. In adding x and y, the $|z|$-th bit of the sum depends whether a *carry* is *generated* or *propagated*. Define the predicates GEN, PROP by having $\text{GEN}(x,y,z)$ hold iff the $|z|$-th bit of both x and y is 1 and $\text{PROP}(x,y,z)$ hold iff the $|z|$-th bit of either x or y is 1. Define $carry(x,y,0) = 0$ and $carry(x,y,s_i(z))$ to be 1 iff

$$(\exists u B z)[\text{GEN}(x,y,u) \wedge (\forall v B z)[|v| > |u| \rightarrow \text{PROP}(x,y,v)]].$$

Then $sum(x,y,z) = x \oplus y \oplus carry(x,y,z)$ where the EXCLUSIVE-OR $x \oplus y$ is defined by $cond(x, cond(y,0,1), cond(y,1,0))$. Using the 2's complement trick, modified subtraction $x \dot{-} y = \max(x-y, 0)$ can be shown to belong to A_0. In order to arithmetize machine computations, pairing and sequence encoding functions are needed. To that end, define the *pairing* function $\tau(x,y)$ by

$$\tau(x,y) = (2^{\max(|x|,|y|)} + x) * (2^{\max(|x|,|y|)} + y). \tag{20}$$

Noting that $2^{\max(|x|,|y|)} = cond(msp(x,y), pad(1,y), pad(1,x))$, this function is easily definable from msp, $cond$, pad, $*$, $+$ hence belongs to A_0. As an example, to compute $\tau(4,3)$, note that $\max(|4|,|3|) = 3$ and so one concatenates $1\underline{100}$ with $1\underline{011}$, where the underlined portions represent 4 resp. 3 in binary. Define the functions TR [resp. TL] which truncate the rightmost [resp. leftmost] bit: $\text{TR}(x) = \text{MSP}(x,1) = \lfloor \frac{x}{2} \rfloor$ and $\text{TL}(x) = \text{LSP}(x, |\text{TR}(x)|) = \text{TR}(rev(\text{TR}(rev(s_1(x)))))$, where the latter definition is used later to show that TL belongs to a certain subclass of A_0. The left π_1 and right π_2 projections are defined by

$$\pi_1(z) = \text{TL}(\text{MSP}(z, \lfloor \frac{|z|}{2} \rfloor)) \tag{21}$$

$$\pi_2(z) = \text{TL}(\text{LSP}(z, \lfloor \frac{|z|}{2} \rfloor)) \tag{22}$$

and satisfy $\tau(\pi_1(z), \pi_2(z)) = z$, $\pi_1(\tau(x,y)) = x$ and $\pi_2(\tau(x,y)) = y$. An n-tuple $(x_1, \ldots, x_n)$ can be encoded by $\tau_n(x_1, \ldots, x_n)$, where $\tau_2 = \tau$ and

$$\tau_{k+1}(x_1, \ldots, x_{k+1}) = \tau(x_1, \tau_k(x_2, \ldots, x_{k+1})).$$

At this point, it should be mentioned that by using the functions so far defined, Turing machine configurations (TM and RATM) are easily expressed in A_0, and even in subalgebras of A_0. A *configuration* of RATM is of the form $(q, u_1, \ldots, u_{k+2}, n_1, \ldots, n_{k+2})$ where $q \in Q$, $u_i \in (\Gamma \cup \{B\})^*$ and $n_i \in \mathbf{N}$. The u_i represent the contents of the k work tapes and of the index query and the index answer tapes, and the n_i represent the head positions on the tapes (the input tape has no head). Since the input is accessed through random access, the input does not form part of the configuration of the RATM. Let ℓ_i [resp. r_i] represent the contents of the left portion [resp. the reverse of the right portion] of the i-th tape (i.e. tape cells of index $\le n_i$ [resp.

$> n_i]$). Assuming some simple binary encoding of $\Gamma \cup \{B\}$, a RATM configuration can be represented using the tupling function by

$$\tau_{2k+5}(q, \ell_1, r_1, \ldots, \ell_{k+2}, r_{k+2}).$$

Let $\text{INITIAL}_M(x)$ be the function mapping x to the initial configuration of RATM M on input x. For configurations α, β in the computation of RATM M on x, let predicate $\text{NEXT}_M(x, \alpha, \beta)$ hold iff $(x, \alpha) \vdash_M (x, \beta)$.

If M is a TM with input x, then a configuration can be similarly represented by $\tau_{2k+3}(q, \ell_0, r_0, \ldots, \ell_k, r_k)$ where $initial_M(x)$, $next_M(x, \alpha, \beta)$ are the counterparts for Turing machine computations without random access.

Lemma 3.11 INITIAL_M, NEXT_M *belong to* $[0, I, s_0, s_1, \text{BIT}, |x|; \text{COMP}, \text{CRN}]$. *Moreover,* τ, π_1, π_2, $initial_M$, $next_M$ *belong to* $[0, I, s_0, s_1, \text{MOD2}, msp; \text{COMP}, \text{CRN}]$.

Proof. Using s_0, s_1, *pad*, $*$, $\lfloor x/2 \rfloor$, *cond*, BIT, MSP, LSP, the pairing and tupling functions, etc. it is routine to show that INITIAL_M, NEXT_M are definable in A_0 without use of the smash function. For instance, a move of the first tape head to the right would mean that in the next configuration $\ell_1' = 2 \cdot \ell_1 + \text{MOD2}(r_1)$ and $r_1' = \lfloor r_1/2 \rfloor$.

Temporarily, let $\mathcal{F}$ designate the algebra $[0, I, s_0, s_1, \text{MOD2}, msp; \text{COMP}, \text{CRN}]$. Using MOD2 and msp appropriately, functions from (2) through (14) can be introduced in $\mathcal{F}$. For instance, in (2)

$$rev0(x, s_i(y)) = s_{\text{MOD2}(msp(x,y))}(rev0(x, y)).$$

Part-of quantifiers, the pairing function (20), its left, right projections (21) can be defined in $\mathcal{F}$, by using msp, lsp appropriately in place of MSP, LSP. For instance, to define the projections of the pairing function, define auxiliary functions g, h as follows:

$$\begin{aligned} g(0, x) &= 0 \\ g(s_i(z), x) &= s_{\text{BIT}(z*z, ones(x))}(g(z, x)) \\ h(x) &= rev(g(x, x)). \end{aligned}$$

Then $|h(x)| = \lfloor \frac{|x|}{2} \rfloor$ and for x of even length (i.e. $ones(h(x)) * ones(h(x)) = ones(x)$), the left and right projections of the pairing function are defined by

$$\begin{aligned} \pi_1(x) &= msp(x, h(x)) \\ \pi_2(x) &= lsp(x, h(x)). \end{aligned}$$

From this, the function $initial_M$ and predicate $next_M$ are now routine to define. ∎

We can now describe how short sequences of small numbers are encoded in A_0. To illustrate the idea, what follows is a first approximation to the sequence encoding technique. Generalizing the pairing function, to encode the sequence (3,9,0,4) first compute $\max\{|3|, |9|, |0|, |4|\}$. Temporarily let t denote the integer having binary representation

$$1\underline{0011}1\underline{1001}1\underline{0000}1\underline{0100}$$

where the underlined portions correspond to the binary representations of 3,9,0,4. Now the length ℓ of sequence $(3, 9, 0, 4)$ is 4, the *block size* BS is 5, and $|t| = \ell \cdot \text{BS}$. Define, as a first approximation, the *sequence number* $\langle 3, 9, 0, 4 \rangle$ by $\tau(t, \ell)$.

Given the sequence number $z = \langle 3, 9, 0, 4 \rangle$, the Gödel β function decoding the sequence is given by

$$\beta(0, z) = \pi_2(z) = \ell = 4.$$

The blocksize $\text{BS} = \lfloor |\pi_1(z)|/\pi_2(z) \rfloor = \lfloor 20/4 \rfloor = 5$, and for $i = 1, \ldots, 4$

$$\beta(i, z) = \text{LSP}(\text{MSP}(\pi_1(z), (\ell - i) \cdot \text{BS}), \text{BS} - 1).$$

Thus $\beta(1, z) = \text{LSP}(\text{MSP}(\pi_1(z), 3 \cdot 5), 4) = 3$, etc. All the above operations belong to A_0, with the exception of multiplication and division (which provably do not belong to A_0). However, multiplication and division by powers of 2 is possible in A_0, so the previously described sequence encoding technique is slightly modified. The sequence $(a_1, \ldots, a_n)$ is encoded by $z = \langle a_1, \ldots, a_n \rangle$ where

$$\begin{aligned} & z = \tau(t, n) \\ & \text{BS} = \max\{2^{||a_i||} : 1 \leq i \leq n\} \\ & t = h(N) \end{aligned}$$

where

$$\begin{aligned} & |N| = n \cdot \text{BS} \\ & h(0) = 0 \\ & h(s_i(x)) = s_{g(x)}(h(x)) \end{aligned}$$

and

$$g(x) = \begin{cases} 1 & \text{if } |x| \bmod \text{BS} = 0 \\ \text{BIT}((\text{BS} \dot{-} 1) \dot{-} (|x| \bmod \text{BS}), a_{\lfloor |x|/\text{BS} \rfloor + 1}) & \text{else.} \end{cases}$$

Finally define

(23) $$\ell h(z) = \beta(0, z) = \begin{cases} \pi_2(z) & \text{if } z \text{ encodes a pair} \\ 0 & \text{else} \end{cases}$$

and for $1 \leq i \leq \beta(0, z)$

(24)

$$\beta(i, z) = \text{LSP}(\text{MSP}(\pi_1(z), (\ell h(z) \dot{-} i) \cdot \lfloor \frac{|\pi_1(z)|}{\ell h(z)} \rfloor), \lfloor \frac{|\pi_1(z)|}{\ell h(z)} \rfloor \dot{-} 1).$$

Suppose that $z = \tau(t, n)$ codes a sequence of length n, where $|t| = \text{BS} \cdot n$ and the block size $\text{BS} = 2^m$ for some m. The exponent m can be computed, since $m = \mu x \leq ||a|| \quad [\text{MSP}(|t|, x) = n]$, and A_0 is closed under sharply bounded minimization. Using this observation, it is clear that the β function belongs to A_0. Using the techniques introduced, the following can be proved.

Theorem 3.12 (Clote [30]) *If $f \in A_0$ then there exists $g \in A_0$ such that for all x,*

$$g(x, \vec{y}) = \langle f(0, \vec{y}), \ldots, f(|x| - 1, \vec{y}) \rangle.$$

The following two lemmas, together with the sequence encoding machinery of A_0, will allow us soon to establish that $A_0 = \mathcal{F}\text{LH}$.

Lemma 3.13 *For every $k, m > 1$,*

$$\text{DTIMESPACE}(\log^k(n), \log^{1-1/m}(n)) \subseteq A_0.$$

Proof. Let M be a RATM running in time $\log^k(n)$ and space $\log^{1-1/m}(n)$. For each $i \leq m \cdot k$, define a predicate $\text{NEXT}_{M,i}$ belonging to A_0 such that

$$\text{NEXT}_{M,i}(x,c,d) \iff d \text{ follows } c \text{ in at most } \log^{i/m}(n) \text{ steps} \tag{25}$$

where c, d are encodings of configurations in the computation of M on input x, and $n = |x|$. By Lemma 3.11, the predicate $\text{NEXT}_{M,0}$ belongs to A_0 and satisfies (25). Assume that $\text{NEXT}_{M,i} \in A_0$ has been defined and satisfies (25). Define the formula $\text{NEXT}_{M,i+1}(x,c,d)$ by

$$\begin{aligned}(\exists s \leq |x|^3)(\forall j < ||x||^{1/m} - 1) \quad & [s = \langle s_0, \ldots, s_{||x||^{1/m}-1}\rangle \wedge \\ & c = s_0 \wedge d = s_{||x||^{1/m}-1} \wedge \text{NEXT}_{M,i}(s_j, s_{j+1})].\end{aligned} \tag{26}$$

Since for all $j < ||x||^{1/m}$, $|s_j| \leq ||x||^{1-1/m}$, $|s| \leq (||x||^{1-1/m}+1) \cdot ||x||^{1/m} \leq 2 \cdot ||x||$. This establishes the validity of the bound $s \leq |x|^3$ in the definition of $\text{NEXT}_{M,i+1}$. It follows that M accepts input x iff $\text{NEXT}_{M,m\cdot k}(x,c,d)$ holds, where c and d respectively are the initial configuration and the terminal accepting configuration in the computation of M on x. ∎

The following result for LH is similar.

Lemma 3.14 *DSPACE*$(\log\log(n))$ *on a* RATM *is contained in* LH.

Proof. Using the logtime computable pairing function from Fact 2.4, one can define a logtime predicate $\text{NEXT}_{M,0}$ which identifies consecutive configurations in the computation of the RATM M running in polylogarithmic time ($\log^{O(1)}(n)$) and simultaneous sublogarithmic space ($\log^{1-\epsilon}(n)$). As in the preceding lemma, by using ATM existential and universal branching, the predicate $\text{NEXT}_{M,i}$ can be shown to belong to LH. Thus $\text{DTIMESPACE}(\log^k(n), \log^{1-1/m}(n)) \subseteq$ LH. Since there are only $2^{c\cdot\log\log n} = \log^c(n)$ many possible configurations for some constant c, it follows that DSPACE$(O(\log\log(n)))$ on RATM is contained in

$$\text{DTIMESPACE}(\log^k(n), \log^{1-1/m}(n))$$

on RATM, hence in LH. ∎

Theorem 3.15 (P. Clote) $A_0 = \mathcal{F}\text{LH}$.

Proof. Consider the direction $A_0 \subseteq \mathcal{F}\text{LH}$. It follows from Fact 2.4 that 0, s_0, s_1, $|x|$ are computable in logtime. To compute $\text{BIT}(i,x)$, the machine M_1 on input $BiBxB$ writes the bits of i onto a work tape, computes $|i|$ and writes $i + |i| + 1$ onto its input query tape and reads the input answer tape. To compute the i-th bit of $I_k^n(x_1, \ldots, x_n)$, the machine M_2 on input $BiBx_1Bx_2B\cdots Bx_nB$ uses existential and universal states find the locations of the input separators B, computes $m = i + k + \sum_{j<i}|x_j|$ and returns the m-th bit of the input. To compute the the i-th bit of $x\#y = 2^{|x|\cdot|y|}$, the machine M_3 outputs 1 if $i = |x| \cdot |y|$, else 0. Since the product $|x| \cdot |y|$ can be computed in DSPACE$(\log\log n)$ on a RATM, hence in LH by Lemma 3.14, it follows that $\# \in \mathcal{F}\text{LH}$.

To see that $\mathcal{F}\text{LH}$ is closed under composition, suppose that $f(x)$ equals $g(h_1(x), h_2(x))$, where the bitgraphs of g, h_1, h_2 are computed by the ATM M_g, M_{h_1}, M_{h_2} running in logtime with constantly many alternations. The bitgraph of f is then computed by the ATM M_f obtained from M_g as follows. Recall that M_g expects input of the form By_1By_2B, where $y_1, y_2 \in \{0,1\}^*$. Whenever M_g requests the i-th bit of its input, M_f computes $|h_1(x)|$, $|h_2(x)|$, and then executes the following code.

```
if i = 0 then
     return B
else if i ≤ |h_1(x)| then
     return M_{h_1}(i − 1, x)
else if i = |h_1(x)| + 1 then
     return B
else if i ≤ |h_1(x)| + 1 + |h_2(x)| then
     return M_{h_2}(i − |h_1(x)| − 2, x)
else
     return B
```

Inequalities like $i \leq |h_1(x)|$ can be decided by checking whether $|i| < ||h_1(x)||$ or $|i| = ||h_1(x)||$ and i precedes $|h_1(x)|$ in lexicographic order (i.e. $iB|h_1(x)|$). Values $M_{h_1}(i-1,x)$ can be computed by simulating M_{h_1}, providing bits of input $i-1$ when required, etc. It is similarly easy to see that $\mathcal{F}$LH is closed under CRN. It follows that $A_0 \subseteq \mathcal{F}$LH.

Consider the direction that $\mathcal{F}\text{LH} \subseteq A_0$. A first attempt to arithmetize the computation of the logtime bounded RATM M on input x might be to use Lemma 3.11 together with sequence numbers. However, this encoding of M's computation cannot be done in A_0 because there are $O(\log n)$ many configurations in M's computation, with each configuration of size $O(\log n)$, thus requiring sequence numbers of size $O(\log^2 n)$, and quantification over such values is not sharply bounded. However, the integer s, which encodes the sequence of *instructions* executed (rather than configurations), *is* bounded by a polynomial in n and so can be expressed within the scope of a sharply bounded quantifier. What then remains to be shown is the existence of functions in A_0 which recognize whether a sequence of instructions corresponds to a correct computation.

Suppose that $M = (Q, \Sigma, \Gamma, \Delta, q_0, k+2, \ell)$, is a Σ_m-RATM, running in time $c \cdot |n|$, where $n = |x|$. For notational simplicity, assume $c = 1$ and that $\Sigma = \{0, 1\}$. An instruction of M is of the form

$$(q, a_1, \ldots, a_{k+2}, q', b_1, \ldots, b_{k+2}, d_1, \ldots, d_{k+2})$$

belonging to the transition relation Δ, where q is the current state, $a_1, \ldots, a_{k+2} \in (\Gamma \cup \{B\})$ are the symbols currently read on the k work tapes and input query and answer tapes, q' is the next state, $b_1, \ldots, b_{k+2} \in (\Gamma \cup \{B\})$ are the symbols printed on the work tapes and input query and answer tapes, and $d_i \in \{-1, 0, 1\}$ is the direction of head movement on tape $1 \leq i \leq k+2$.

Using the earlier sequence encoding, one can code the sequence of $\log n$ instructions by an integer bounded by $|x|^{O(1)}$. Thus the Σ_m machine M accepts input x iff

$$(\exists y_1 \leq |x|^d)(\forall y_2 \leq |x|^d)(\exists y_3 \leq |x|^d) \cdots (Q y_m \leq |x|^d)\Theta(x, y_1, \ldots, y_m),$$

where Q is $\forall$ (resp. $\exists$) if m is even (resp. odd) and $\Theta(x, \vec{y})$ says that

if

> *(i)* for $i = 1, 2, \ldots, m$, each y_i encodes a sequence of instructions from M's program, where the states occurring in y_i are existential (resp. universal) if i is odd (resp. even),

and

> *(ii)* the sequence of instructions coded by $y_1, \ldots, y_m$ determines a correct computation of M,

then

(iii) this computation is accepting.

Using BIT, MSP, LSP, etc. it is not difficult to express *(i)* and *(iii)* in A_0. It remains to see how to formulate *(ii)* in A_0. If $y_1, \ldots, y_m$ encodes a sequence s of $m \cdot \log n$ instructions from M's program, then s corresponds to a correct computation of M, provided that

- The state in the first instruction is q_0, the state in the last instruction is q_A, and for all $0 \leq r < m \cdot \log n - 1$, the new state in the r-th instruction is the old state in the $r+1$-st instruction.
- For all tape cells, and all $r < m \cdot \log n$, if the r-th instruction is

$$(q, a_1, \ldots, a_{k+2}, q', b_1, \ldots, b_{k+2}, d_1 \ldots, d_{k+2})$$

  then for $1 \leq j \leq k+2$, a_j is the symbol written on the j-th work tape at the last visit of the position p_j, where p_j is the current head position of the j-th work tape, provided that this position has previously been visited, and $a_j = B$ otherwise. Moreover, if q is the input query state q_I, then $b_{k+2} = \text{BIT}(i, x)$, where i is the content of the input query tape.

Note that the function SBBITSUM (sharply bounded bitsum)

$$\text{SBBITSUM}(x, y) = \begin{cases} \sum_{i<|y|} \text{BIT}(i, y) & \text{if } y \leq |x| \\ |x| + 1 & \text{else} \end{cases}$$

is computable in $\log^2 n$ time and $\log\log$ space, hence by Lemma 3.13, belongs to the algebra A_0. Using SBBITSUM, one can determine whether, given i_0, i_1, j, at instruction i_0 the head of tape j is in the same position as at instruction i_1 in the execution of M on input x. It follows that $\mathcal{F}\text{LH} \subseteq A_0$. ∎

In Furst et al. [51], integer multiplication was shown to be AC^0 reducible to MAJ, where $\text{MAJ}(x)$ is 1 if $\sum_{i<|x|} \text{BIT}(i, x) \geq \lceil |x|/2 \rceil$, else 0. In Chandra et al. [25] as refined by Barrington et al. [6], MAJ was shown to be AC^0 reducible to integer multiplication. The following characterization of polysize, constant depth threshold circuits TC^0 is proved by formalizing these reductions, using the previous techniques.

Theorem 3.16 (Clote-Takeuti [37])

$$\text{TC}^0 \quad = \quad [0, I, s_0, s_1, |x|, \text{BIT}, \times, \#; \text{COMP}, \text{CRN}].$$

Remark 3.17 Theorem 3.15 was first obtained by combining the author's result [39] that A_0 equals FO definable functions, and the Barrington-Immerman-Straubing result [6] that FO = LH, an analogue of Bennett's Theorem 3.56.[11] The current proof is direct, influenced by A. Woods' presentation in [144], and simplifies the argument of [6] by using Lemma 3.13 and Lemma 3.14, both of which were generalized from Lemma 3.55. Lemma 3.55 was first proved by Nepomnjascii [101] (a related result proved by Bennett [13]), though R. Kannan [79] later rediscovered this result. The idea of encoding a sequence of instructions rather than a sequence of configurations has been repeatedly used by a number of persons.

[11] See [6] for definition of FO.

3.2 Bounded recursion on notation

Cobham's original characterization of $\mathcal{F}$PTIME was in terms of functions on words in a finite alphabet, rather than integers. To our knowledge, the first published proof of Cobham's result, additionally formulated for functions on the integers, appeared in [117].

Definition 3.18 The function f is defined by *bounded recursion on notation* (BRN) from g, h_0, h_1, k if

$$\begin{array}{rcl} f(0,\vec{y}) & = & g(\vec{y}), \\ f(s_0(x),\vec{y}) & = & h_0(x,\vec{y},f(x,\vec{y})), \text{ if } x \neq 0 \\ f(s_1(x),\vec{y}) & = & h_1(x,\vec{y},f(x,\vec{y})) \end{array}$$

provided that $f(x,\vec{y}) \leq k(x,\vec{y})$ for all $x,\vec{y}$.

Theorem 3.19 (A. Cobham [40], see H. Rose [117])

$$\mathcal{F}\text{PTIME} = [0, I, s_0, s_1, \#; \text{COMP}, \text{BRN}].$$

Proof. Temporarily denote the algebra $[0, I, s_0, s_1, \#; \text{COMP}, \text{BRN}]$ by $\mathcal{F}$. Consider first the inclusion from left to right. Let M be a TM with input x, running in polynomial time $p(|x|)$. Using BRN the functions MOD2,TR,msp can be defined in $\mathcal{F}$ as follows: $\text{MOD2}(0) = 0$, $\text{MOD2}(s_0(x)) = 0$, $\text{MOD2}(s_1(x)) = 1$; $\text{TR}(0) = 0$, $\text{TR}(s_i(x)) = x$; $msp(x,0) = x$, $msp(x,s_i(y)) = \text{TR}(msp(x,y))$, where $\text{MOD2}(x)$, $\text{TR}(x)$, $msp(x,y)$ are bounded by x. It follows that

$$[0, I, s_0, s_1, \text{MOD2}, msp; \text{COMP}, \text{CRN}] \subseteq \mathcal{F}$$

so by Lemma 3.11, $initial_M$, $next_M$ belong to $\mathcal{F}$. By suitably composing $0, s_0, s_1, \#$, there is a function $k \in \mathcal{F}$ satisfying $p(|x|) \leq |k(x)|$ for all inputs x. Using BRN, define

$$\begin{array}{rcl} Run_M(x,0) & = & initial_M(x) \\ Run_M(x,s_i(y)) & = & next_M(x, Run_M(x,y)). \end{array}$$

Then the value computed by M on input x can be obtained from $Run_M(x,k(x))$ by π_1, π_2.

The inclusion from right to left is proved by an easy induction on term formation in the Cobham algebra. ■

Using the same techniques, one can characterize the class $\mathcal{G}\text{TIMESPACE}(n^{O(1)}, O(n))$ of polynomial time linear space computable functions of linear growth as follows. The first assertion is due to D.B. Thompson [135] (recall that $*$ is concatenation), and the other assertion follows by an alternate function in bounding the recursion on notation.

Theorem 3.20

$$\begin{array}{rcl} \mathcal{G}\text{TIMESPACE}(n^{O(1)}, O(n)) & = & [0, I, s_0, s_1, *; \text{COMP}, \text{BRN}] \\ & = & [0, I, s_0, s_1, \times; \text{COMP}, \text{BRN}]. \end{array}$$

Definition 3.21 The function f is defined from functions g, h_0, h_1, k by *sharply bounded recursion on notation* [12] (SBRN) if

$$\begin{aligned} f(0,\vec{y}) &= g(\vec{y}) \\ f(s_0(x),\vec{y}) &= h_0(x,\vec{y},f(x,\vec{y})), \text{ if } x \neq 0 \\ f(s_1(x),\vec{y}) &= h_1(x,\vec{y},f(x,\vec{y})), \end{aligned}$$

provided that $f(x,\vec{y}) \leq |k(x,\vec{y})|$ for all $x, \vec{y}$.

In [94], J. Lind characterized $\mathcal{F}$LOGSPACE functions on words $w \in \Sigma^*$ as the smallest class of functions containing the initial functions $c_=$ (characteristic function of equality), $*$ (string concatenation) and closed under the operations of explicit transformation, log bounded recursion on notation, and a (provably stronger) version of concatenation on notation. An arithmetic version of Lind's characterization is the following.

Theorem 3.22

$$\begin{aligned} \mathcal{F}\text{LOGSPACE} &= [0, I, s_0, s_1, |x|, \text{BIT}, \#; \text{COMP}, \text{CRN}, \text{SBRN}] \\ &= [0, I, s_0, s_1, \text{MOD2}, msp, \#; \text{COMP}, \text{CRN}, \text{SBRN}]. \end{aligned}$$

The first statement appeared in [38, 37] and the second can be proved using similar techniques.

Recently, function algebras have been found for small parallel complexity classes. Consider the following variants of recursion on notation.

Definition 3.23 The function f is defined by *k-bounded recursion on notation* ($k -$ BRN) from g, h_0, h_1 if

$$\begin{aligned} f(0,\vec{y}) &= g(\vec{y}) \\ f(s_0(x),\vec{y}) &= h_0(x,\vec{y},f(x,\vec{y})), \text{ if } x \neq 0 \\ f(s_1(x),\vec{y}) &= h_1(x,\vec{y},f(x,\vec{y})) \end{aligned}$$

provided that $f(x,\vec{y}) \leq k$ holds for all $x, \vec{y}$, where k is a constant.

Definition 3.24 The function f is defined by *weak bounded recursion on notation* (WBRN) from g, h_0, h_1, k if $F(x,\vec{y})$ is defined from g, h_0, h_1, k by BRN and $f(x,\vec{y}) = F(|x|,\vec{y})$; i.e.

$$\begin{aligned} F(0,\vec{y}) &= g(\vec{y}) \\ F(s_0(x),\vec{y}) &= h_0(x,\vec{y},F(x,\vec{y})), \text{ if } x \neq 0 \\ F(s_1(x),\vec{y}) &= h_1(x,\vec{y},F(x,\vec{y})) \\ f(x,\vec{y}) &= F(|x|,\vec{y}) \end{aligned}$$

provided that $F(x,\vec{y}) \leq k(x,\vec{y})$ holds for all $x, \vec{y}$.

The characterization of polynomial size, constant depth boolean circuits with parity gates (resp. MOD6 gates) uses sequence encoding techniques of A_0 together with logtime hierarchy analogues of work of Handley, Paris, Wilkie [63].

Theorem 3.25 (Clote-Takeuti [37])

$$\text{ACC}(2) = [0, I, s_0, s_1, |x|, \text{BIT}, \#; \text{COMP}, \text{CRN}, 1 - \text{BRN}] \tag{27}$$

$$\text{ACC}(6) = [0, I, s_0, s_1, |x|, \text{BIT}, \#; \text{COMP}, \text{CRN}, 2 - \text{BRN}] \tag{28}$$

$$\text{ACC}(6) = [0, I, s_0, s_1, |x|, \text{BIT}, \#; \text{COMP}, \text{CRN}, 3 - \text{BRN}]. \tag{29}$$

[12] In [37], this scheme was denoted B_2RN.

The following characterization of $\mathcal{F}$ALOGTIME uses earlier techniques with a formalization of Barrington's trick [7] in Theorem 2.10 of expressing boolean connectives AND, OR by permutation group word problems.

Theorem 3.26 (P. Clote [30])

$$\mathcal{F}\text{ALOGTIME} = [0, I, s_0, s_1, |x|, \text{BIT}, \#; \text{COMP}, \text{CRN}, 4 - \text{BRN}].$$

A natural question arising from work in vectorizing compilers is whether there is a recursive procedure to effectively *parallelize* sequential code (i.e. from sequential code, generate optimal code for a parallel machine). Though I am not aware of details having been worked out, it seems clear that the non-existence of such a procedure must follow from the unsolvability of the halting problem. More importantly, it is not known whether NC is properly contained in PTIME, with modular powering $a^b \bmod m$ being a candidate to separate the classes. Though effective optimal parallelization of sequential code is hopeless, it may seem surprising that certain well-known parallel complexity classes can be characterized in a sequential manner. From the following theorem it follows that NC is characterized by a fragment of the PASCAL language allowing only for-loops of the form

```
for i = 1 to |x| if P then y := 2*y else y := 2*y+1;
for i = 1 to ||x|| if <statement>;
```

Using repeated squaring (see proof of Theorem 3.53), modular powering is evidently a polynomial time algorithm, yet cannot obviously be written using the above two for-loops.

Theorem 3.27 (P. Clote [39])

$$\begin{aligned} \text{NC} &= [0, I, s_0, s_1, |x|, \text{BIT}, \#; \text{COMP}, \text{CRN}, \text{WBRN}] \\ \text{AC}^k &= \{f \in \text{NC} : rk_{\text{WBRN}}(f) \leq k\}. \end{aligned}$$

It should be mentioned that independently and at about the same time, B. Allen [3] characterized NC by a function algebra using a form of *divide and conquer recursion*, and noticed without giving details that over a basis of appropriate initial functions, NC could also be characterized by the scheme of WBRN.[13] A precise statement of Allen's characterization is given later in Theorem 3.77.

Using such techniques, two characterizations of NC^k were given in [39, 37]. Levels of a natural time-space hierarchy between $\mathcal{F}$PTIME and $\mathcal{F}$PSPACE were characterized in [29].

3.3 Bounded recursion

In 1953, A. Grzegorczyk [58] investigated a hierarchy of subclasses $\mathcal{E}^n$ of primitive recursive functions, defined as the closure of certain initial functions under composition and bounded recursion.

Definition 3.28 The function f is defined by *bounded recursion* (BR) from functions g, h, k if

$$\begin{aligned} f(0, \vec{y}) &= g(\vec{y}) \\ f(x+1, \vec{y}) &= h(x, \vec{y}, f(x, \vec{y})) \end{aligned}$$

provided that $f(x, \vec{y}) \leq k(x, \vec{y})$ holds for all $x, \vec{y}$.

[13] See remark at bottom of p. 13 of [3].

Definition 3.29 Define the following *principal* functions

$$
\begin{aligned}
f_0(x) &= s(x) = x+1 \\
f_1(x,y) &= x+y \\
f_2(x,y) &= (x+1)\cdot(y+1) \\
f_3(x) &= 2^x \\
f_{n+1}(x) &= f_n^{(x)}(1), \text{ for } n \geq 3
\end{aligned}
$$

Let $\mathcal{E}f$ denote $[0, I, s, f; \text{COMP}, \text{BR}]$ and $\mathcal{E}^n$ denote $\mathcal{E}f_n$.

Remark 3.30 For $n \geq 3$, Grzegorczyk's [58] original functions were defined by $f_{n+1}(0,y) = f_n(y+1, y+1)$, and $f_{n+1}(x+1,y) = f_{n+1}(x, f_{n+1}(x,y))$. The above functions, for $n \geq 3$, were taken from [12].

A number of characterizations of Grzegorczyk's classes $\mathcal{E}^n$ $\mathcal{E}^n$, for $n \geq 3$, have been given. A. Meyer and D. Ritchie [97] characterized $\mathcal{E}^n$ in terms of certain *loop* programming languages, H. Schwichtenberg [123] investigated the number of nested bounded recursions used in function definitions (the so-called *Heinermann hierarchy*), S.S. Muchnick [99] investigated *vectorized* Grzegorczyk classes (essentially related to simultaneous bounded recursion schemes), etc. The following theorem is due to H. Schwichtenberg [123] for $n \geq 3$ and to H. Müller [100] for $n = 2$.[14]

Theorem 3.31 (Schwichtenberg [123], Müller [100]) *Let $\mathcal{H}_n$ be the set*

$$\{f : f \textit{ primitive recursive, } rk_{\text{PR}}(f) \leq n\}.$$

Then for $n \geq 2$, $\mathcal{H}_n = \mathcal{E}^{n+1}$.

In [58] Grzegorczyk proved that for all $n \geq 0$, $\mathcal{E}^n$ is properly contained in $\mathcal{E}^{n+1}$ by demonstrating that $f_{n+1} \notin \mathcal{E}^n$. Concerning the relational classes, he showed that for $n \geq 2$, $\mathcal{E}^n_*$ is properly contained in $\mathcal{E}^{n+1}_*$, and asked whether $\mathcal{E}^0_* \subset \mathcal{E}^1_* \subset \mathcal{E}^2_*$. This question remains open. In fact, $\text{LTH} \subseteq \mathcal{E}^0_*$ and $\mathcal{E}^2_* = \text{LINSPACE}$,[15] so Grzegorczyk's question is related to the yet open problem whether the linear time hierarchy is properly contained in linear space. An interesting partial result concerning the containment of the first two relational classes is the following.

Theorem 3.32 (A. Bel'tyukov [12]) *For $s \geq 1$, let $\beta_s(x) = \max(1, x + \lceil x^{1-1/s} \rceil)$. Then for $s \geq 1$, $\mathcal{E}^0_* = (\mathcal{E}\beta_s)_*$. Additionally, $\mathcal{E}^2_* = \mathcal{E}^1_*$ implies $\mathcal{E}^2_* = \mathcal{E}^0_*$*

To obtain this result, Bel'tyukov introduced the *stack register machine*, a machine model capable of describing $(\mathcal{E}f)_*$. The stack register machine, a variant of the successor register machine, has a finite number of *input registers* and *stack* registers $S_0, \ldots, S_k$ together with a *work* register W. Branching instructions

if $p(x_1, \ldots, x_m) = q(x_1, \ldots, x_m)$ **then** I_i **else** I_j

allow to jump to different instructions I_i, I_j depending on the comparison of two polynomials whose variables are current register values. Storage instructions

W = S_i

allow a value to be saved from a stack register to the work register. Incremental instructions

[14] It should be mentioned that [123] used slightly different functions f_i; there f_i is the i-th Ackermann branch A_i.

[15] See Corollary 3.37.

```
Si = Si + 1
```

perform the only computation, and have a side effect of setting to 0 all S_j for $j < i$. A program is a finite list of instructions, where for each i there is at most one incremental instruction for S_i.

Apart from characterizing $\mathcal{E}^2_*$ or LINSPACE, Bel'tyukov characterized the linear time hierarchy LTH. The papers of Paris, Wilkie [110] and Handley, Paris, Wilkie [63] study counting classes between LTH and LINSPACE defined by stack register machines. Recent work of the author [32] and of W. Handley [65, 64] further study the effect of nondeterminism for this model.

Lemma 3.33 (Grzegorczyk [58]) *The functions $x \dot{-} y$, $sg(x)$, $\overline{sg}(x)$, $sg(x) \cdot y$, $\overline{sg}(x) \cdot y$ belong to $\mathcal{E}^0$. If $f \in \mathcal{E}^0$ then $\sum_{i \leq x} sg(f(i))$, $\sum_{i \leq x} \overline{sg}(f(i))$, $\Pi_{i \leq x} sg(f(i))$ and $\Pi_{i \leq x} \overline{sg}(f(i))$ belong to $\mathcal{E}^0$.*

Definition 3.34 The function f is defined by *bounded minimization* (BMIN) from the function g, denoted by $f(x, \vec{y}) = \mu i \leq x[g(i, \vec{y}) = 0]$, if

$$f(x, \vec{y}) = \begin{cases} \min\{i \leq x : g(i, \vec{y}) = 0\} & \text{if such exists} \\ 0 & \text{else.} \end{cases}$$

Corollary 3.35 (Grzegorczyk [58]) *For $n \geq 0$, $\mathcal{E}^n_*$ is closed under boolean connectives and bounded quantification, and $\mathcal{E}^n$ is closed under bounded minimization.*

Proof. The predicate $\neg P(\vec{x})$ has characteristic function

$$\overline{sg}(c_P(\vec{x})),$$

the predicate $P(\vec{x}) \vee Q(\vec{x})$ has characteristic function

$$\overline{sg}(\overline{sg}(c_P(\vec{x})) \cdot \overline{sg}(c_Q(\vec{x}))),$$

while $(\exists i \leq x) R(i, \vec{y})$ has characteristic function

$$sg(s(x) \dot{-} \sum_{i \leq x} \overline{sg}(c_R(i, \vec{y}))),$$

and $(\forall i \leq x) R(i, \vec{y})$ has characteristic function

$$\overline{sg}(s(x) \dot{-} \sum_{i \leq x} c_R(i, \vec{y})).$$

To define $f(x, \vec{y}) = \mu i \leq x[g(i, \vec{y}) = 0]$, define the auxiliary function h by

$$\begin{aligned} h(i, \vec{y}) &= \sum_{j \leq i} \overline{sg}(g(j, \vec{y})) \\ &= \text{cardinality of } \{j \leq i : g(j, \vec{y}) = 0\} \end{aligned}$$

so $\overline{sg}(h(i, \vec{y})) = 1$ iff $(\forall j \leq i)(g(j, \vec{y}) \neq 0)$, and

$$\sum_{i \leq x} \overline{sg}(h(i, \vec{y})) = \begin{cases} \mu i \leq x[g(i, \vec{y}) = 0] & \text{if } (\exists i \leq x)(g(i, \vec{y}) = 0) \\ x + 1 & \text{else.} \end{cases}$$

Then

$$f(x, \vec{y}) = \overline{sg}((\sum_{i \leq x} \overline{sg}(h(i, \vec{y}))) \dot{-} x) \cdot \sum_{i \leq x} \overline{sg}(h(i, \vec{y})).$$

∎

The following characterization of LINSPACE in terms of the Grzegorczyk hierarchy was proved by R.W. Ritchie [114].

Theorem 3.36 $\mathcal{F}\text{LINSPACE} = \mathcal{E}^2$.

Proof. Consider first the direction from right to left. The initial functions of $\mathcal{E}^2$ are computable in LINSPACE, and $\mathcal{F}$LINSPACE is closed under composition and bounded recursion.

Now consider the direction from left to right. We first claim that

$$[0, I, s_0, s_1, \text{MOD2}, msp; \text{COMP}, \text{CRN}] \subseteq \mathcal{E}^2.$$

Clearly $s_0, s_1, cond \in \mathcal{E}^2$ and $\mathcal{E}^2$ is closed under bounded quantification. Now $\overline{sg}(0) = 1$, $\overline{sg}(s(x)) = 0$, $\text{MOD2}(0) = 0$, $\text{MOD2}(s(x)) = \overline{sg}(\text{MOD2}(x))$, so that $\text{MOD2} \in \mathcal{E}^2$. Using BR define the following functions in $\mathcal{E}^2$:

$$\begin{aligned}
\lfloor x/2 \rfloor &= \mu y \leq x[y + y = x \vee y + y + 1 = x] \\
\text{MSP}(x, 0) &= x \\
\text{MSP}(x, i+1) &= \lfloor \text{MSP}(x, i)/2 \rfloor \\
\text{BIT}(i, x) &= \text{MOD2}(\text{MSP}(x, i)).
\end{aligned}$$

Temporarily define the auxiliary function h by

$$\begin{aligned}
h(x, 0) &= 0 \\
h(x, i+1) &= \begin{cases} h(x, i) + 1 & \text{if BIT}(i, x) = 1 \\ h(x, i) & \text{else.} \end{cases}
\end{aligned}$$

Note that $ones(x) = 2^{|x|} - 1$ is defined by

$$\mu y \leq s_0(x)[(\forall i \leq x)(\text{BIT}(i, y) = 1 \leftrightarrow (\exists j \leq x)(i \leq j \wedge \text{BIT}(j, x) = 1))].$$

Then $|x| = h(ones(x), x)$ and $msp(x, y) = \text{MSP}(x, |y|)$ belong to $\mathcal{E}^2$.

Suppose that f is defined from g, h_0, h_1 by CRN, where $g, h_0, h_1 \in \mathcal{E}^2$. Then $f(x, \vec{y})$ is $\mu z \leq g(\vec{y}) \cdot (2x + 1) + 2x[(30) \vee (31) \vee (32) \vee (33)]$ where

$$|z| = |g(\vec{y})| + |x| \tag{30}$$

$$\text{MSP}(z, |x|) = g(\vec{y}) \tag{31}$$

$$\begin{gathered}
(\forall i, j < |x|)(j = |x| - i - 1 \wedge \text{BIT}(j, x) = 0 \\
\rightarrow \\
\text{BIT}(j, z) = h_0(\text{MSP}(x, j+1), \vec{y}))
\end{gathered} \tag{32}$$

$$\begin{gathered}
(\forall i, j < |x|)(j = |x| - i - 1 \wedge \text{BIT}(j, x) = 1 \\
\rightarrow \\
\text{BIT}(j, z) = h_1(\text{MSP}(x, j+1), \vec{y})).
\end{gathered} \tag{33}$$

The above bound on f suffices, since $f(x, \vec{y}) \leq g(\vec{y}) \cdot 2^{|x|} + 2^{|x|} - 1$, and the latter is at most $g(\vec{y}) \cdot (2x + 1) + 2x$. By Corollary 3.35, $\mathcal{E}^2$ is closed under BMIN, so $f \in \mathcal{E}^2$. It follows that

$$[0, I, s_0, s_1, \text{MOD2}, msp; \text{COMP}, \text{CRN}] \subseteq \mathcal{E}^2.$$

Now let M be a linear space bounded multitape Turing machine computing a function f. By Lemma 3.11, $initial_M$ and $next_M$ belong to $\mathcal{E}^2$. Define the function T by

$$\begin{aligned}
T(x, 0) &= initial_M(x) \\
T(x, y+1) &= next_M(T(x, y)).
\end{aligned}$$

From the linear space bound, there exists a constant c such that $|T(x,y)| \leq c \cdot |x|$, and so $T(x,y) \leq (x+1)^c + 1$. Thus T is definable using bounded recursion from functions belonging to $\mathcal{E}^2$. It follows that $\mathcal{F}\text{LINSPACE} \subseteq \mathcal{E}^2$. ∎

Corollary 3.37 (R. Ritchie [114]) $\text{LINSPACE} = \mathcal{E}^2_*$.

The following well-known fact follows from Ritchie's arithmetization techniques together with the observations that a TM with space bound $S(n)$ has time bound $2^{O(S(n))}$ and that $2^x \in \mathcal{E}^k$ for $k \geq 3$.

Theorem 3.38 *For $k \geq 3$,*

$$\mathcal{E}^k = \text{DTIME}(\mathcal{E}^k) = \text{DSPACE}(\mathcal{E}^k).$$

Similar techniques yield a characterization of PSPACE.

Theorem 3.39 (D.B. Thompson [135])

$$\mathcal{F}\text{PSPACE} = [0, I, s, \max, x^{|x|}; \text{COMP}, \text{BR}].$$

Definition 3.40 Let k be an integer. The function f is defined by *k-bounded recursion* (k-BR) from functions g, h, k if

$$\begin{aligned} f(0,\vec{y}) &= g(\vec{y}) \\ f(x+1,\vec{y}) &= h(x,\vec{y},f(x,\vec{y})) \end{aligned}$$

provided that $f(x,\vec{y}) \leq k$ holds for all $x, \vec{y}$, where k is a constant.

The following characterization results from the method of the proof of Barrington's Theorem 2.10, arithmetization techniques of this paper, and Theorem 2.12 implying that $\text{ATIME}(n^{O(1)}) = \text{PSPACE}$. In [23], J.-Y. Cai and M. Furst give a related characterization of PSPACE using *safe-storage* Turing machines, a model related to Bel'tyukov's earlier stack register machines. The next result follows from a characterization of PSPACE by a variant of the stack register machine model.

Theorem 3.41 (P. Clote [32]) *For $k \geq 4$,*

$$\text{PSPACE} = [0, I, s_0, s_1, |x|, \text{BIT}, \#; \text{COMP}, \text{CRN}, k\text{-BR}]_*.$$

In [141] K. Wagner extended Ritchie's characterization to more general complexity classes.

Theorem 3.42 (K. Wagner [141]) *Let f be an increasing function such that for some $r > 1$ and for all but finitely many x, it is the case that $f(x) \geq x^r$. Let $\mathcal{F}$ temporarily denote the algebra $[|f(2^n)|; \text{COMP}]$. Then recalling that $f_2(x,y) = (x+1)\cdot(y+1)$,*

$$\text{DSPACE}(\mathcal{F}) = [0, I, s, \max, f; \text{COMP}, \text{BR}]_* = [0, I, s, f_2, f; \text{COMP}, \text{BR}]_*.$$

Let $|x|_0 = x$, $|x|_{k+1} = ||x|_k|$.

Corollary 3.43 (K. Wagner [141]) *For $k \geq 1$,*

$$\text{DSPACE}(n \cdot (\log^{(k)} n)^{O(1)}) = [0, I, s, \max, x^{|x|_{k+1}}; \text{COMP}, \text{BR}]_*.$$

Definition 3.44 (K. Wagner [141]) The function f is defined by *weak bounded primitive recursion* (WBPR) from g, h, k if $f(x, \vec{y}) = F(|x|, \vec{y})$, where F is defined by bounded primitive recursion, i.e.

$$\begin{aligned} F(0, \vec{y}) &= g(\vec{y}) \\ F(x+1, \vec{y}) &= h(x, \vec{y}, F(x, \vec{y})) \\ f(x, \vec{y}) &= F(|x|, \vec{y}) \end{aligned}$$

provided that $F(x, \vec{y}) \leq k(x, \vec{y})$ holds for all $x, \vec{y}$.

Provided the proper initial functions are chosen, WBPR is equivalent with BRN. Using this observation, Wagner characterized certain general complexity classes as follows.

Theorem 3.45 (K. Wagner [141]) *Let f be an increasing function such that $f(x) \geq x^r$ for some $r > 1$, and temporarily let $\mathcal{F}$ denote the algebra $[|f(2^n)|; \text{COMP}]$ and $\mathcal{G}$ denote $\{g^k : g \in \mathcal{F}, k \in \mathbf{N}\}$. Then*

$$\begin{aligned} \text{DTIMESPACE}(\mathcal{G}, \mathcal{F}) &= [0, I, s_0, s_1, \max, f; \text{COMP}, \text{BRN}]_* \\ &= [0, I, s, \max, 2 \cdot x, f; \text{COMP}, \text{BRN}]_* \\ &= [0, I, s, +, f; \text{COMP}, \text{BRN}]_* \\ &= [0, I, s, \max, 2 \cdot x, \lfloor x/2 \rfloor, \dot{-}, f; \text{COMP}, \text{WBPR}]_* \\ &= [0, I, s, \lfloor x/2 \rfloor, +, \dot{-}, f; \text{COMP}, \text{WBPR}]_*. \end{aligned}$$

The class DTIMESPACE$(n^{O(1)}, O(n))$ of simultaneous polynomial time and linear space can be characterized from the previous theorem by taking $f(x) = x^2$. As referenced in [142], S.V. Pakhomov [109] has characterized general complexity classes DTIMESPACE(T, S), DTIME(T), and DSPACE(S) for suitable classes S, T of unary functions. The class QL = DTIME$(n \cdot (\log n)^{O(1)})$ of *quasilinear time* was studied by C.P. Schnorr in [121]. In analogy, let *quasilinear space* be the class DSPACE$(n \cdot (\log n)^{O(1)})$. Though Corollary 3.43 characterizes quasilinear space via a function algebra, there appears to be no known function algebra for *quasilinear time*. In [61], Y. Gurevich and S. Shelah studied the class NLT (*nearly linear time*) of functions computable in time $O(n \cdot (\log n)^{O(1)})$ on a random access Turing machine RTM, which is allowed to change its input tape.

Definition 3.46 (Gurevich-Shelah [61]) A RTM is a Turing machine with one-way infinite main tape, address tape and auxiliary tape, such that the head of the main tape is at all times in the cell whose position is given by the contents of the address tape. Instructions of a RTM are of the form

$$(q, a_0, a_1, a_2) \rightarrow (q', b_0, b_1, b_2, d_1, d_2)$$

where q, q' are states, $a_i, b_i \in (\Sigma \cup \{B\})$, and $d_i \in \{-1, +1\}$. For such an instruction, if the machine is in state q reading a_0, a_1, a_2 on the main, address and auxiliary tapes, then the machine goes to state q', writes b_0, b_1, b_2 on the respective tapes, and the head of the address [resp. auxiliary] tape goes one square to the right if $d_1 = 1$ [resp. $d_2 = 1$] otherwise one square to the left.

In [61], Gurevich and Shelah show the robustness of NLT by proving the equivalence of this class with respect to different machine models, and give a function algebra for NLT. Their algebra, defined over words from a finite alphabet, is the closure under composition of certain initial functions and weak iterates $f^{(|x|)}(x)$ of certain string manipulating initial functions.

3.4 Bounded minimization

In [58], Grzegorczyk considered function classes defined by bounded minimization, defined in Definition 3.34.

3.47. Definition ([68, 69]). For $n = 0, 1, 2$, define $\mathcal{M}^n = [0, I, s, f_n; \text{COMP}, \text{BMIN}]$. For $n \geq 3$, define $\mathcal{M}^n = [0, I, s, x^y, f_n; \text{COMP}, \text{BMIN}]$.

Though implicitly asserted in [58], the details for the proof of the following result, which follow those in the proof of Theorem 3.65, are given by K. Harrow [69].[16] The idea of the proof is simply to encode via sequence numbers a definition by bounded primitive recursion and apply the bounded minimization operator.

Theorem 3.48 (Grzegorczyk [58], Harrow [69]) *For $n \geq 3$, $\mathcal{E}^n = \mathcal{M}^n$.*

In the literature, the algebra RF of *rudimentary functions* is sometimes defined by

$$\text{RF} = [0, I, s, +, \times; \text{COMP}, \text{BMIN}].$$

As noticed in [69], it follows from J. Robinson's [115] bounded quantifier definition of addition from successor and multiplication that $\mathcal{M}^2 = \text{RF}$. As is well-known, there is a close relationship between (bounded) minimization and (bounded) quantification. This is formalized as follows.

Terms in the first order language of $0, s, +, \cdot, \leq$ of arithmetic are defined inductively by: 0 is a term; $x_0, x_1, \ldots$ are terms; if t, t' are terms, then $s(t)$, $t + t'$ and $t \cdot t'$ are terms. Atomic formulas are of the form $t = t'$ and $t \leq t'$, where t, t' are terms. The set Δ_0 of bounded quantifier formulas is defined inductively by: if ϕ is an atomic formula, then $\phi \in \Delta_0$; if $\phi, \theta \in \Delta_0$ then $\neg\phi$, $\phi \wedge \theta$, and $\phi \vee \theta$ belong to Δ_0; if $\phi \in \Delta_0$ and t is a term, then $(\exists x \leq t)\phi(x, t)$ and $(\forall x \leq t)\phi(x, t)$ belong to Δ_0. A k-ary relation $R \subseteq \mathbf{N}^k$ belongs to Δ_0^N if there is a Δ_0 formula ϕ for which $R = \{(a_1, \ldots, a_n) : \mathbf{N} \models \phi(a_1, \ldots, a_n)\}$.

Definition 3.49 A predicate $R \subseteq \mathbf{N}^k$ belongs to CA (*constructive arithmetic*), a notion due to R. Smullyan, if there is $\phi(\vec{x}) \in \Delta_0$ such that $R(a_1, \ldots, a_k)$ holds iff $\mathbf{N} \models \phi(a_1, \ldots, a_k)$. Following Definition 2.13, a function $f(\vec{x}) \in \mathcal{G}\text{CA}$ if the bitgraph $B_f \in \text{CA}$ and f is of linear growth.[17]

Definition 3.50 *Presburger arithmetic* (PRES) is the collection of all predicates $R \subseteq \mathbf{N}^k$ for which there exists a first order formula $\phi(\vec{x})$ in the language $0, s, +$ of arithmetic such that $R(a_1, \ldots, a_k)$ holds iff $\mathbf{N} \models \phi(a_1, \ldots, a_k)$.

The following theorem is proved by using quantifier elimination for Presburger arithmetic to show the equivalence between first order formulas and bounded formulas in a richer language allowing congruences, and then exploiting the correspondence between bounded quantification and bounded minimization.

Theorem 3.51 (Harrow [68]) *$\mathcal{M}^1_*$ equals the collection of Presburger definable sets* .

From J. Robinson's definition of addition from successor and multiplication, the following easily follows.

Proposition 3.52 (Harrow [68]) $\mathcal{M}^2_* = \text{CA}$ *and* $\mathcal{M}^2 = \mathcal{G}\text{CA}$.

[16] The result is proved in [69] for $[0, I, s, x^y, f_n; \text{COMP}, \text{BMIN}]$, but as, previously explained, the exponential is unnecessary.

[17] In the literature, especially in [110], a function f is defined to be Δ_0^N if its graph G_f belongs to Δ_0^N and f is of linear growth. It easily follows from Corollary 3.54 that $f \in \mathcal{G}\text{CA} \iff f \in \Delta_0^N$.

While it is obvious that $\text{RF}_* = \text{CA}$, it is non-trivial and surprising that CA equals the linear time hierarchy . In [13], J.H. Bennett showed that the collection of constructive arithmetic sets (Δ_0 definable) is equal to RUD, the class of *rudimentary* sets in the sense of [132]. Later, C. Wrathall [145] proved that the rudimentary sets are exactly those in the linear time hierarchy LTH.

Theorem 3.53 (J. Bennett [13]) *The ternary relation $G(x, y, z)$ for the graph $x^y = z$ of exponentiation is in constructive arithmetic.*

Proof. Using the technique of *repeated squaring* to compute the exponential $x^y = z$, the idea is to encode the computation in a Δ_0 manner where all quantifiers are bounded by a polynomial in z. Suppose that $x, y > 1$ and that $y = \sum_{i<n} y_i \cdot 2^i$ is the binary representation of y. The following algorithm computes $z = x^y$ by repeatedly applying the fact that $x^{2y} = (x^y)^2$ and $x^{2y+1} = (x^y)^2 \cdot x$. Throughout the rest of the proof, let n denote $|y|$.

```
z = 1
for i = 1 to n
    if y_{n-i} = 0 then z = z^2 else z = z^2· x
```

To encode the computation, for $0 \le i \le n$, let $a_i = \text{MSP}(y, |y| - i)$ and $b_i = x^{a_i}$. The binary representation of a_i consists of the i leftmost bits of y, while b_i equals the value of z at the end of the i-th pass through the above **for**-loop. Except for trivial cases where x, y take values among $0, 1$ it follows that $x^y = z$ if and only if there exist sequences $(a_0, \ldots, a_n)$ and $(b_0, \ldots, b_n)$ for which

$$a_0 = 0, b_0 = 1, a_n = y, b_n = z, (\forall i < n)(a_{i+1} \in \{2a_i, 2a_i + 1\})$$

and

$$(\forall i < n)((a_{i+1} = 0 \rightarrow b_{i+1} = b_i^2) \wedge (a_{i+1} = 1 \rightarrow b_{i+1} = b_i^2 \cdot x)).$$

Thus the graph of exponentiation is Δ_0 provided that sequences $(a_0, \ldots, a_n)$, $(b_0, \ldots, b_n)$ can be encoded in a manner where all quantifiers are bounded by a polynomial in z.

To do this, we will find relatively prime $m_0 < m_1 < \cdots < m_n$ satisfying $a_i, b_i < m_i$ and apply the Chinese remainder theorem to obtain $M = \Pi_{i \le n} m_i$ and unique $A, B < M$ for which

$$\begin{aligned} A &\equiv a_i \pmod{m_i} \\ B &\equiv b_i \pmod{m_i} \end{aligned}$$

for $i \le n$. By choosing the m_i to be prime powers of distinct primes, and m_{i+1} to be the smallest prime power divisor of M greater than m_i, one can determine the m_i from M in a Δ_0 manner. Formally, define the predicates $\text{PRIME}(p)$, $\text{MPP}(m, M)$, $\text{I}(m, M)$, $\text{N}(m, m', M)$, and $\text{F}(m, M)$ as follows.

1. $\text{PRIME}(p)$ means that p is prime:

$$2 \le p \wedge (\forall q < p)(q | p \rightarrow q = 1).$$

2. $\text{MPP}(m, M)$ means that m is a maximal prime power divisor of M:

$$m | M \wedge (\exists p \le m)(\text{PRIME}(p) \wedge p | m \wedge (\forall q < m)(q | m \rightarrow q \in \{1, p\}) \wedge p \cdot m \not| M).$$

3. $\text{I}(m, M)$ means that $m = m_0$ is the *initial* (least) maximal prime power divisor of M:

$$(m = 1 \wedge M \in \{0, 1\}) \vee (\text{MPP}(m, M) \wedge (\forall m' \le M)(\text{MPP}(m', M) \rightarrow m \le m').$$

4. $\text{N}(m, m', M)$ means that $m' = m_{i+1}$ is the *next* maximal prime power divisor of M after $m = m_i$:

$$\text{MPP}(m, M) \wedge \text{MPP}(m', M) \wedge m < m' \wedge (\forall m'' < m')(\text{MPP}(m'', M) \rightarrow m'' \leq m).$$

5. $\text{F}(m, M)$ means that $m = m_n$ is the *final* (greatest) maximal prime power divisor of M:

$$(m = 1 \wedge M \in \{0, 1\}) \vee (\text{MPP}(m, M) \wedge (\forall m' \leq M)(\text{MPP}(m', M) \rightarrow m' \leq m).$$

Assume that neither x nor y take values among $0, 1$. Then $a_i < 2^i$ and $b_i \leq x^{2^i}$ for $i \leq n$. Define $m_0, \ldots, m_n$ to be an increasing sequence of prime powers of distinct primes as follows. Let $m_0 = 2$. Given $m_0, \ldots, m_{k-1}$ let $m_k = p^\alpha$, where p is the least prime not dividing $\Pi_{i<k} m_i$ and α is the least integer for which $p^\alpha > x^{2^k}$. By the prime number theorem, $p = O(k \cdot \ln k) < k^2 \leq x^{2^k}$, and by choice of α, it is the case that $p^{\alpha-1} < x^{2^k}$, and so

$$p^\alpha = p \cdot p^{\alpha-1} < x^{2^k} \cdot x^{2^k} \leq x^{2^{k+1}}.$$

An inductive proof yields that $\Pi_{i<k} m_i \leq x^{2^{k+1}}$ for all $k > 0$, hence

$$M = \Pi_{i \leq n} m_i \leq x^{2^{n+2}} = (x^{2^{n-1}})^8 \leq z^8.$$

It now follows that the relation $x^y = z$ is Δ_0 definable. ∎

The main lines of this proof were influenced by Wilkie's presentation in [143]. See [62] for other proofs.

Corollary 3.54 *The function algebra* $[0, I, s_0, s_1, |x|, \text{BIT}; \text{COMP}, \text{CRN}]$ *is contained in* $\mathcal{M}^2$.

Proof. Note that $s_0(x) = x + x$, $s_1(x) = x + x + 1$,

$$|x| = \mu y \leq x[(\exists z \leq 2 \cdot x)(2^y = z \wedge x < z \wedge \lfloor z/2 \rfloor \leq x)]$$

and that

$$\text{BIT}(i, x) = \mu y \leq 1[(\exists u, v \leq x)(|u| = i + 1 \wedge v = 2^{i+1} \wedge v | (x - u))]$$

so that $s_0, s_1, |x|, \text{BIT}$ belong to $\mathcal{M}^2$. Using these functions and bounded minimization, it is easy to show that $\mathcal{M}^2$ is closed under CRN. ∎

The following is proved in a manner similar to that of Lemma 3.13 and Lemma 3.14.

Lemma 3.55 (Nepomnjascii [101]) *For every* $k, m > 1$, $\text{NTIMESPACE}(n^k, n^{1-1/m})$ *on a* TM *is contained in* CA. *Moreover,* $\text{NSPACE}(O(\log(n))) \subseteq \text{LTH}$.

Theorem 3.56 LTH = CA.

Proof. Consider first the direction from left to right. It follows from Lemmas 3.54 and 3.11 that $initial_M$ and $next_M$ are CA. Now proceed in a similar fashion as in the proof of Theorem 3.15.

The direction from right to left is proved by induction on the length of Δ_0 formulas. Addition, inequality $\leq$, and multiplication are computable in LOGSPACE, and $\mathcal{F}$LOGSPACE is closed under composition. By Lemma 3.55 it follows that atomic Δ_0 formulas define relations in LTH. Bounded quantifiers can be handled by existential and universal branching of an alternating Turing machine. ∎

Corollary 3.57 $\mathcal{M}_*^2 = \text{LTH}$, *and* $\mathcal{M}^2 = \mathcal{F}\text{LTH}$.

Though the linear time hierarchy equals the bounded arithmetic hierarchy, there is no known exact level-by-level result. The sharpest result we know is due to A. Woods [144].

If Γ is a class of first order formulas, then Γ^N denotes the collection of predicates definable by a formula in Γ. Let $\Sigma_{0,m}$ denote the collection of bounded quantifier formulas of the form $(\exists \vec{x}_1 \leq y)(\forall \vec{x}_2 \leq y) \ldots (Q\vec{x}_m \leq y)\phi$ where ϕ is a quantifier free formula in the first order language $0, 1, +, \cdot, \leq$. Thus $\Sigma_{0,0}$ is the collection of quantifier free formulas. In the following theorem, recall the definition of $\Sigma_n - \text{TIME}(T(n))$ from Definition 2.8.

Theorem 3.58 (A. Woods [144]) *For* $m \geq 1$, $\Sigma_{0,m}^N \subseteq \Sigma_{m+2} - \text{TIME}(n)$.

Sketch of Proof. The inclusion $\Sigma_{0,0}^N \subseteq \Sigma_2 - \text{TIME}(O(n))$ is shown as follows. Given an atomic formula $\phi(n)$, suppose that all terms appearing in $\phi(n)$ are bounded by a polynomial in n. By the prime number theorem, there exists a constant c such that the product of the first $c \cdot \ln(n)$ primes is larger than the values of all terms occurring in $\phi(n)$. Using non-determinism guess all terms and subterms appearing in the given quantifier free formula, guess the first $c \cdot \ln(n)$ many prime numbers p and the residues modulo p of the products of subterms occurring in a term, and branching universally, verify that the computations are correct. Now, by the Chinese remainder theorem, the computations are correct iff they are correct modulo the primes.

Since the negation of a quantifier free formula is quantifier free, it follows that

$$\Sigma_{0,0}^N \subseteq \Sigma_2 - \text{TIME}(O(n)) \cap \Pi_2 - \text{TIME}(O(n)).$$

Now induct on the number of quantifier blocks. ∎

By Corollary 3.57 and Theorem 3.36, $\mathcal{M}_*^2 = \text{LTH} \subseteq \text{LINSPACE} = \mathcal{E}_*^2$. While LINSPACE is clearly closed under *counting*, this may not be the case for LTH. A typical open question is whether $\pi(x) \in \mathcal{M}^2$, where $\pi(x)$ is the number of primes less than x. In [110, 63], J. Paris, A. Wilkie and later W. Handley studied the effect of adding k-bounded recursion to LTH. Using the techniques of Barrington, Paris, Wilkie and Handley, together with those of this paper, the following result can be proved.

Theorem 3.59 (P. Clote [32])
For any $k \geq 4$, $\text{ALINTIME} = [0, I, s, +, \dot{-}, \times; \text{COMP}, \text{BMIN}, k\text{-BR}]_*$.

As in Corollary 3.57, $\mathcal{F}\text{PH}$ can similarly be characterized.

Theorem 3.60 (Folklore)

$$\begin{aligned} \mathcal{F}\text{PH} &= [0, I, s, +, \dot{-}, \times, \#; \text{COMP}, \text{BMIN}] \\ &= [0, I, s, +, \dot{-}, \times, \#; \text{COMP}, \text{BRN}, \text{BMIN}] \\ &= [0, I, s_0, s_1, \#; \text{COMP}, \text{BRN}, \text{BMIN}]. \end{aligned}$$

The last assertion of this theorem was sharpened by S. Bellantoni as follows. Following S. Buss [18] let $\Box_i^P$ denote the class of functions computed in polynomial time on a Turing machine with oracle A, for some set $A \in \Sigma_i^P$. With this notation, $\mathcal{F}\text{PH} = \cup_i \Box_i^P$.

Definition 3.61 (S. Bellantoni [10]) Let μFP_i denote the algebra

$$\{f \in [0, I, s_0, s_1, \#; \text{COMP}, \text{BRN}, \text{BMIN}] : rk_{\text{BMIN}}(f) \leq i\}.$$

Theorem 3.62 (S. Bellantoni [10]) *For* $i \geq 0$, $\Box_i^P = \mu FP_i$.

3.5 Divide and conquer, course-of-values and miscellaneous

Definition 3.63 (Grzegorczyk [58], Constable [41])
The function f is defined by *bounded summation* (BSUM) [resp. *bounded product* (BPROD)] from g, k if $f(x, \vec{y})$ equals

$$\Sigma_{i=0}^{x} g(i, \vec{y}) \qquad \text{[resp. } \Pi_{i=0}^{x} g(i, \vec{y})\text{]}$$

provided that $f(x, \vec{y}) \leq k(x, \vec{y})$ for all $x, \vec{y}$.

The function f is defined by *sharply bounded summation* (SBSUM) [resp. *sharply bounded product* (SBPROD)] from g, k if $f(x, \vec{y})$ equals

$$\Sigma_{i=0}^{|x|} g(i, \vec{y}) \qquad \text{[resp. } \Pi_{i=0}^{|x|} g(i, \vec{y})\text{]}$$

provided that $f(x, \vec{y}) \leq k(x, \vec{y})$ for all $x, \vec{y}$.[18]

The elementary functions were first introduced by Kalmár [78] and Csillag [46].

Definition 3.64 The class $\mathcal{E}$ of *elementary* functions is the algebra

$$[0, I, s, +, \dot{-}; \text{COMP}, \text{BSUM}, \text{BPROD}].$$

The elementary functions have many alternate characterizations, among them that $\mathcal{E} = \mathcal{E}^3$.

Theorem 3.65 (Grzegorczyk [58])

$$\begin{aligned} \mathcal{E} &= [0, I, s, f_3; \text{COMP}, \text{BR}] \\ &= [0, I, s, \dot{-}, x^y; \text{COMP}, \text{BMIN}] \\ &= [0, I, s, \dot{-}, \times, x^y; \text{COMP}, \text{BSUM}]. \end{aligned}$$

Grzegorczyk asked whether $\mathcal{E}$ had a *finite basis*, i.e. a finite number of functions, whose closure with I under composition equals $\mathcal{E}$. As surveyed in [142], D. Rödding first gave a positive answer, which was refined by C. Parsons. In [95], S.S. Marčenkov gave a particularly elegant characterization of $\mathcal{E}$ as $[0, I, s, \lfloor x/y \rfloor, x^y, \phi(x, y); \text{COMP}]$, where $\phi(x, y)$ is 0 for $x \leq 1$, and otherwise is the least index i for which $a_i = 0$ in the radix x representation of y, i.e. $y = \sum_{i=0}^{\infty} a_i \cdot x^i$, where $0 \leq a_i < x$ for all i. In the following theorem, the first statement is due to S.S. Marčenkov [95], while the second statement to J.P. Jones and Y. Matijasevič [76].

Theorem 3.66

$$\begin{aligned} \mathcal{E}^3_* &= ([0, I, s, \dot{-}, \lfloor x/y \rfloor, x^y; \text{COMP}])_* \\ &= ([0, I, +, \dot{-}, \lfloor x/y \rfloor, x!, 2^x; \text{COMP}])_*. \end{aligned}$$

In [41], R. Constable defined the class $\mathcal{K}$ by

$$[0, I, s, +, \dot{-}, \times, \lfloor x/y \rfloor; \text{COMP}, \text{SBSUM}, \text{SBPROD}]$$

a polynomial analogue of the definition of Kalmár elementary functions. The class $\mathcal{K}(f)$ is defined as above, but with f as an additional initial function. Let $FP(f)$ denote the collection of functions polynomial time computable in f ($FP(f)$ can equivalently be defined as the set of type 1 functions in $BFF(f)$; see Definition 4.6).

On p. 118 of [41], the following claim is stated as a theorem without proof.

[18] Sharply bounded summation [resp. product] is called *weak sum* [resp. *product* in [142], and bounded summation [resp. product] is called *limited sum* [resp. *product* in [58].

Claim 3.67 For all non-decreasing f, $K(f) = FP(f)$.

The statement $\mathcal{F}\text{PTIME} = \mathcal{K}$ was then claimed as a corollary in [41]. This statement was again cited as a theorem (without proof) in [142].

It now appears that this assertion is doubtful, since $\mathcal{K} \subseteq NC$ and it is currently conjectured that NC is properly contained in $\mathcal{F}\text{PTIME}$. Moreover, using an oracle separation of NC^A from P^A, the author [31] provided a counterexample to the previous claim.

Theorem 3.68 (P. Clote [31]) *There exists a non-decreasing function f for which $K(f) \neq FP(f)$.*

Nevertheless, Constable's class $\mathcal{K}$ is very natural, suggesting the following question.

Question 3.69 *What complexity class corresponds to*

$$[0, I, s, +, \dot{-}, \times, \lfloor x/y \rfloor; \text{COMP}, \text{SBSUM}, \text{SBPROD}]?$$

H.-J. Burtschick (personal correspondence) suggested that polynomial size uniform arithmetic circuits could be related to the class $\mathcal{K}$.

Somewhat related is the recent work on counting classes. The class $\#P$, introduced by Valiant [138], is the set of functions f, for which there exists a nondeterministic polynomial time bounded Turing machine M, such that $f(x)$ is the number of accepting paths in the computation tree of M on input x. Unless $P = NP$, it is unlikely that $\#P$ is closed under composition. Using the arithmetization of boolean formulas from A. Shamir (see [5]), H. Vollmer and K. Wagner gave the following characterization of $\#P$.[19]

Theorem 3.70 (H. Vollmer, K. Wagner [140])

$$\begin{aligned} \#P &= [[0, I, S, +, \dot{-}, \times, \lfloor x/y \rfloor, \#; \text{COMP}, \text{SBPROD}]; \text{BSUM}] \\ &= [[0, I, S, +, \dot{-}, \times, \#; \text{COMP}]; \text{SBPROD}, \text{BSUM}]. \end{aligned}$$

Definition 3.71 Let $\mathcal{R}_k$ be the smallest class of functions definable from the constant functions $0, \ldots, k$, the projections I, the characteristic functions of the *graphs* of $+, \times, =$, and closed under composition and bounded recursion.

The following result was proved by the Paris-Wilkie modification of Bel'tyukov's stack register machines.

Theorem 3.72 (Paris, Wilkie [110]) $(\mathcal{R}_2)_* = (\mathcal{R}_3)_*$.

The next theorem follows from the author's work in [32] and is based on Barrington's trick.

Theorem 3.73 (P. Clote [32]) *For $n \geq 4$,*

$$(\mathcal{R}_n)_* = (\mathcal{R}_{n+1})_* = \text{ALINTIME}.$$

In [87], Kutyłowski considered oracle versions of the Paris-Wilkie work.

[19] In the notation of [140], the characterization reads $\#P = \left[[+, \dot{-}, \times, :]_{\text{Sub, WProd}}\right]_{\text{Sum}}$, and $\#P = \left[[+, \dot{-}, \times]_{\text{Sub}}\right]_{\text{WProd,Sum}}$. This formulation is equivalent to that given in Theorem 3.70.

Definition 3.74 (M. Kutyłowski [87]) f is a k-function[20] if for all $x_1, \ldots, x_n$

$$f(x_1, \cdots, x_n) = f(\min(x_1, k), \cdots, \min(x_n, k)) \leq k.$$

For a family $\mathcal{F}$ of functions, $\mathcal{W}_k(\mathcal{F})$ is the smallest class of functions containing I, $\mathcal{F}$, all k-functions and closed under composition and k-bounded recursion. The function f is defined from g, h by *m-counting* if

$$\begin{aligned} f(0, \vec{x}) &= g(\vec{x}) \\ f(n+1, \vec{x}) &= (f(n, \vec{x}) + h(n, \vec{x})) \pmod{m} \end{aligned}$$

The class $\mathcal{CW}_k(\mathcal{F})$ is the smallest class of functions containing I, $\mathcal{F}$, all m-functions for $m \in \mathbf{N}$ and closed under composition, k-bounded recursion and arbitrary counting.

Theorem 3.75 (M. Kutyłowski [87]) *For every class $\mathcal{F}$ of functions, $\mathcal{W}_2(\mathcal{F})_* = \mathcal{W}_3(\mathcal{F})_*$. For every $k \geq 3$, there exists a family $\mathcal{F}$ of functions, such that $\mathcal{W}_k(\mathcal{F})_* \subset \mathcal{W}_{k+1}(\mathcal{F})_*$. For every $k \geq 3$, there is a family $\mathcal{F}$ of functions such that $\mathcal{CW}_k(\mathcal{F})_* \subset \mathcal{CW}_{k+1}(\mathcal{F}_*)$.*

Parallel algorithms often employ a divide and conquer strategy. B. Allen [3] formalized this approach to characterize NC.

Definition 3.76 The *front half* $\text{FH}(x)$ is defined by $\text{MSP}(x, \lfloor |x|/2 \rfloor)$ and the *back half* $\text{BH}(x)$ by $\text{LSP}(x, \lfloor |x|/2 \rfloor)$. The function f is defined by *polynomially bounded branching recursion* (PBBR) from functions g, h if there exists a polynomial p such that

$$\begin{aligned} f(0, \vec{y}) &= g_0(\vec{y}) \\ f(1, \vec{y}) &= g_1(vecy) \\ f(x, \vec{y}) &= h(x, \vec{y}, f(\text{FH}(x), \vec{y}), f(\text{BH}(x), \vec{y})), \text{if } x > 1 \end{aligned}$$

provided that $|f(x, \vec{y})| \leq p(\max(|x|, |y_i|))$ for all $x, \vec{y}$. Let $Seq(x) = 0$ if x encodes a sequence[21] else 0. If x encodes a sequence $(x_1, \ldots, x_n)$ and f is a one-place function, then the operation MAP is defined by $\text{MAP}(f, x) = \langle f(x_1), \ldots, f(x_n) \rangle$. Define the *bounded shift left function* by $\text{SHL}(x, i, y) = x \cdot 2^{\min(i, |y|)}$.

Theorem 3.77 (B. Allen [3]) NC *is characterized by the function algebra*

$$[0, I, s, +, \dot{-}, |x|, \text{BIT}, cond, c_{\leq}, Seq, \beta, \text{MSP}, \text{SHL}; \text{COMP}, \text{MAP}, \text{PBBR}].$$

Allen explicitly did not attempt to find the smallest set of initial functions, but went on to develop a proof theory for NC functions, similar in spirit to that of S. Buss [18]. Independently and at the same time, an equivalent theory of arithmetic for NC functions was given by the author [27], later appearing in joint work with G. Takeuti [36].

In [112] F. Pitt considered a variant of B. Allen's polynomial bounded branching recursion, where the function value is bounded by a constant.

[20] What is here called a k-function is called a $k+1$-function in [87]. As our definition of k-bounded recursion corresponds to Kutyłowski's definition of $k+1$-bounded recursion, the indices of $\mathcal{W}_k(\mathcal{F})$ and $\mathcal{CW}_k(\mathcal{F})$ differ by 1 from [87].

[21] Here, we use the earlier defined sequence numbers, though Allen [3] uses a different sequence encoding technique.

Definition 3.78 (F. Pitt [112]) The function f is defined by *k-bounded tree recursion on notation* (k-BTRN) from functions g, h if

$$\begin{array}{rcl} f(0,\vec{y}) & = & g_0(\vec{y}) \\ f(1,\vec{y}) & = & g_1(\vec{y}) \\ f(x,\vec{y}) & = & h(x,\vec{y},f(\text{FH}(x),\vec{y}),f(\text{BH}(x),\vec{y})), \text{if } x>1 \end{array}$$

provided that $f(x,\vec{y}) \leq k$, for all $x, \vec{y}$. When k is unspecified, the scheme k-BTRN is meant to allow all constants $k \in \mathbf{N}$.

Theorem 3.79 (F. Pitt [112])

$$\mathcal{F}\text{ALOGTIME} = [0, I, s_0, s_1, |x|, \#, \text{MSP}, \text{LSP}; \text{COMP}, \text{CRN}, k - \text{BTRN}].$$

The theorem is proved by showing that FH, BH can be defined from the initial functions, and then by defining the function TREE in the above function algebra, where TREE is a function evaluating a full binary tree with alternating levels of AND's and OR's, and whose leaves are the bits of a given input x (see [39, 28] for details of definition). In [39], the author characterized ALOGTIME as $[0, I, s_0, s_1, |x|, \text{BIT}, \#, \text{sc tree}; \text{COMP}, \text{CRN}]$, so the proof sketch is complete.

It is often useful to define two or more functions simultaneously. Simultaneous versions of recursion, recursion on notation, k-bounded recursion on notation, etc. are defined in the obvious manner. For example, simultaneous recursion and simultaneous recursion on notation are defined as follows.

Definition 3.80 The functions $f_1, \ldots, f_n$ are defined from functions $g_1, \ldots, g_n$, $h_1, \ldots, h_n$ by *simultaneous recursion* if

$$\begin{array}{rcl} f_i(0,\vec{y}) & = & g_i(\vec{y}), \text{ for } 1 \leq i \leq n \\ f_i(x+1,\vec{y}) & = & h_i(x,\vec{y},f_1(x,\vec{y}),\ldots,f_n(x,\vec{y})), \text{ for } 1 \leq i \leq n. \end{array}$$

If additionally $f_i(x,\vec{y}) \leq k_i(x,\vec{y})$ for $1 \leq i \leq n$, then the f_i are defined by *simultaneous bounded recursion* from $\vec{g}$, $\vec{h}$, $\vec{k}$.

The functions $f_1, \ldots, f_n$ are defined from functions $g_1, \ldots, g_n$, $h_1^0, \ldots, h_n^0$ and $h_1^1, \ldots, h_n^1$ by *simultaneous recursion on notation* if for $1 \leq i \leq n$

$$\begin{array}{rcl} f_i(0,\vec{y}) & = & g_i(\vec{y}) \\ f_i(s_0(x),\vec{y}) & = & h_i^0(x,\vec{y},f_1(x,\vec{y}),\ldots,f_n(x,\vec{y})), \text{provided } x \neq 0 \\ f_i(s_1(x),\vec{y}) & = & h_i^1(x,\vec{y},f_1(x,\vec{y}),\ldots,f_n(x,\vec{y})). \end{array}$$

If additionally $f_i(x,\vec{y}) \leq k_i(x,\vec{y})$ for $1 \leq i \leq n$, then the f_i are defined by *simultaneous bounded recursion on notation* from $\vec{g}$, $\vec{h^0}$, $\vec{h^1}$, $\vec{k}$.

A function algebra $\mathcal{F}$, whose primary closure operation is a certain form of recursion, can often be proved to be closed under the simultaneous version of that form of recursion, by using the pairing function τ and its projections π_1, π_2. For instance, the following is straightforward to establish.

Proposition 3.81 (Kapron-Cook [80])
The Cobham algebra $[0, I, s_0, s_1, \#; \text{COMP}, \text{BRN}]$ *is closed under simultaneous bounded recursion on notation.*

Proof. For notational simplicity, suppose that $n = 2$. Define

$$\begin{array}{rcl} f(0,\vec{y}) & = & \tau(g_1(\vec{y}),g_2(\vec{y})) \\ f(s_i(x),\vec{y}) & = & \tau(h_1^i(x,\vec{y},\pi_1(f(x,\vec{y})),\pi_2(f(x,\vec{y}))),h_2^i(x,\vec{y},\pi_1(f(x,\vec{y})),\pi_2(f(x,\vec{y})))) \end{array}$$

where $f(x,\vec{y}) \leq \tau(k_1(x,\vec{y}),k_2(x,\vec{y}))$. Then $f_1(x,\vec{y})$ is $\pi_1(f(x,\vec{y}))$ and $f_2(x,\vec{y})$ is $\pi_2(f(x,\vec{y}))$. ∎

The Fibonacci sequence $1, 1, 2, 3, 5, 8, \ldots$ is defined by $Fib(0) = Fib(1) = 1$, and $Fib(n+2) = Fib(n) + Fib(n+1)$. This is a special case of course-of-values recursion.

Definition 3.82 The function f is defined from functions g, h by *course-of-values recursion* (VR) if

$$\begin{aligned} f(0, \vec{y}) &= g(\vec{y}) \\ f(x+1, \vec{y}) &= h(x, \vec{y}, \langle f(0, \vec{y}), \ldots, f(x, \vec{y}) \rangle). \end{aligned}$$

The class $\mathcal{PR}$ of primitive recursive functions is easily seen to be closed under VR.

Definition 3.83 The function f is defined from functions g, h, r, k by *bounded 2-value recursion* (BVR) if

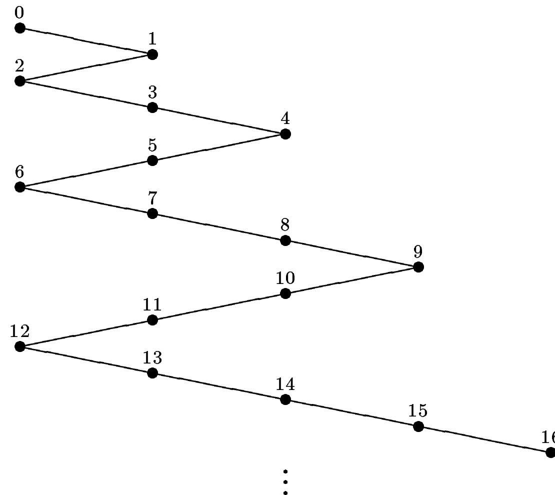
$$\begin{aligned} f(0, \vec{y}) &= g(\vec{y}) \\ f(x+1, \vec{y}) &= h(x, \vec{y}, f(x, \vec{y}), f(r(x, \vec{y}), \vec{y})) \end{aligned}$$

provided that $f(x, \vec{y}) \leq k(x, \vec{y})$ and $r(x, \vec{y}) < x$ for all $x, \vec{y}$.

Theorem 3.84 (Monien [98]) *Let* $f_2(x, y) = (x+1) \cdot (y+1)$. *Then*

$$\{f \in \text{ETIME} : f \text{ has linear growth rate}\} = [0, I, s, f_2; \text{COMP}, \text{BVR}].$$

Proof. Our exposition follows [142]. Temporarily, let $\mathcal{F}$ denote $\{f \in \text{ETIME} : f \text{ has linear growth rate}\}$ and $\mathcal{G}$ denote $[0, I, s, f_2; \text{COMP}, \text{BVR}]$. Consider first the inclusion $\mathcal{F} \subseteq \mathcal{G}$. Suppose that M is a TM which computes the bitgraph B_f of $f : \mathbf{N}^k \to \mathbf{N}$ in time $2^{c \cdot n}$. For notational simplicity, suppose $k = 1$ and $|f(x)| \leq d \cdot |x|$.
Claim $B_f \in \mathcal{G}$.
Since $\sum_{i < 2^{c \cdot n}} i \leq 2^{2c \cdot n}$, without loss of generality, M's head movements before halting may be assumed to be of the form



For notational simplicity, assume that M has only one tape, and that the transition function

$$\delta : (Q - \{q_A, q_R\}) \times (\Sigma \cup \{B\}) \to Q \times (\Sigma \cup \{B\}) \times \{-1, 0, 1\}$$

satisfies $\delta(q,\sigma) = (state(q,\sigma), symbol(q,\sigma), direction(q,\sigma)$ for suitable functions *state*, *symbol*, *direction*. Let $h(t)$ be M's head position at the *beginning* of step t; let $s(t,x)$ be the state of M at the *completion* of step t on input x; let $a(t,x)$ be the symbol written by M on cell $h(t)$ during step t. Let

$$p_0(t,t') = \begin{cases} \max\{t'' : t'' \le t' \wedge h(t'') = h(t)\} & \text{if such exists} \\ t'+1 & \text{else} \end{cases}$$

and $p(t) = p_0(t, t \dot{-} 1)$. Let $sqrt(x) = \lfloor \sqrt{x} \rfloor$. Note that

$$h(t) = \begin{cases} sqrt(x) + sqrt(x)^2 \dot{-} x & \text{if } x \le sqrt(x)^2 + sqrt(x) \\ x \dot{-} sqrt(x) \dot{-} sqrt(x)^2 & \text{else.} \end{cases}$$

Using BMIN, *sqrt* is definable by $sqrt(x) = \mu y \le x[x < (y+1)^2]$ and so $p, h \in \mathcal{M}^2 \subseteq \mathcal{G}$. Define the functions $s(t,x)$ and $a(t,x)$ by

$$\begin{aligned} s(0,x) &= state(q_0, B) \\ s(t+1,x) &= \begin{cases} state(s(t,x), a(p(t+1),x)) & \text{if } p(t+1) \le t \\ state(s(t,x), \text{BIT}(|x| \dot{-} h(t+1), x)) & \text{else and } 1 \le h(t+1) \le |x| \\ state(s(t,x), B) & \text{else} \end{cases} \end{aligned}$$

$$\begin{aligned} a(0,x) &= symbol(q_0, B) \\ a(t+1,x) &= \begin{cases} symbol(s(t,x), a(p(t+1),x)) & \text{if } p(t+1) \le t \\ symbol(s(t,x), \text{BIT}(|x| \dot{-} h(t+1), x)) & \text{else and } 1 \le h(t+1) \le |x| \\ symbol(s(t,x), B) & \text{else.} \end{cases} \end{aligned}$$

Instead of defining the functions $s(t,x)$, $a(t,x)$ by simultaneous bounded recursion, define $F(t,x) = \tau(s(t,x), a(t,x))$ by bounded 2-value recursion in the obvious manner. Since $\tau, \pi_1, \pi_2 \in \mathcal{M}^2 \subseteq \mathcal{G}$, it is now routine to complete the proof of the claim that $B_f \in \mathcal{G}$.

Define

$$f(x) = \mu y \le (x+1)^{d+1} \cdot 2^d[(\forall i \le d \cdot |x|)((x,i) \in B_f \leftrightarrow \text{BIT}(i,y) = 1)].$$

Since BR is included in BVR, by the proof of Theorem 3.36, BIT $\in \mathcal{G}$. By Corollary 3.35, $\mathcal{G}$ is closed under bounded quantification and bounded minimization, so it follows that $f \in \mathcal{G}$.

Consider now the inclusion $\mathcal{G} \subseteq \mathcal{F}$. By induction, all functions of $\mathcal{G}$ are of linear growth rate. The functions $0, I_k^n, s, f_2$ are computable in exponential time, and because functions of $\mathcal{F}$ are of linear growth rate, $\mathcal{F}$ is closed under composition. If f is defined by BVR from g, h, r, k, then when computing $f(x+1, \vec{y})$, an exponential time bounded machine M has sufficient space to store the entire sequence $f(0,\vec{y}), \ldots, f(x,\vec{y})$ of previous values on a work tape. It follows that any function of the algebra $\mathcal{G}$ is computable in exponential time. ■

A more powerful version of simultaneous recursion was introduced in [80].

Definition 3.85 The functions $f_1, \ldots, f_n$ are defined from functions $g_1, \ldots, g_n$, $h_1^0, \ldots, h_n^0$, $h_1^1, \ldots, h_n^1$ and $k_1, \ldots, k_n$ by *multiple bounded recursion on notation* if the f_i are defined by simultaneous recursion on notation from $\vec{g_i}$, $\vec{h_i^0}$, $\vec{h_i^1}$ and moreover

$$\begin{aligned} f_1(x,\vec{y}) &\le k_1(x,\vec{y}) \\ f_i(x,\vec{y}) &\le k_i(x,\vec{y}, f_1(x,\vec{y}), \ldots, f_{i-1}(x,\vec{y})), \text{ for } 2 \le i \le n. \end{aligned}$$

The following non-trivial closure property has an important application in the Kapron-Cook characterization of type 2 polynomial time computations described in the next section.

Theorem 3.86 (Kapron-Cook [80]) *The Cobham algebra* $[0, I, s_0, s_1, \#; \text{COMP}, \text{BRN}]$ *is closed under multiple bounded recursion on notation.*

3.6 Safe recursion

All the function algebras from the previous subsection are defined from specific initial functions, using some version of bounded recursion. Without any bound, even schemes such as WBRN can generate all the primitive recursive functions. Recently, certain *unbounded* recursion schemes have been introduced which distinguish between variables as to their position in a function $f(x_1, \ldots, x_n; y_1, \ldots, y_m)$. Variables x_i occurring to the left of the semi-colon are called *normal*, while variables y_j to the right are called *safe*. By allowing only recursions of a certain form, which distinguish between normal and safe variables, particular complexity classes can be characterized. *Normal* values are considered as known in totality, while *safe* values are those obtained by impredicative means (i.e. via recursion). Sometimes, to help distinguish normal from safe positions, the letters $u, v, w, x, y, z, \ldots$ denote normal variables, while $a, b, c, \ldots$ denote safe variables. This terminology, due to Bellantoni-Cook [11], was chosen to indicate that a *safe* position is one where it is safe to substitute an impredicative value. Related *tiering* notions, though technically different, have occurred in the literature, as in the author's work with G. Takeuti [35] (k sorted variables used in defining k-fold multiple exponential time), but most especially in H. Simmons [130] (*control* variables, i.e. those used for recursion, are distinguished from usual variables; by separating their function, one prevents diagonalization as in the Ackermann function) and in D. Leivant [88, 89, 90, 91] (stratified polymorphism, second order system $L_2(QF^+)$ corresponding to polynomial time computable functions, stratified functional programs, ramified recurrence over 2 tiered word algebras corresponding to polynomial time). Of these, [130] and [89] are the most related to the Bellantoni-Cook work described below.

If $\mathcal{F}$ and $\mathcal{O}$ are collections of initial functions and operations which distinguish normal and safe variables, then $\text{NORMAL} \cap [\mathcal{F}; \mathcal{O}]$ denotes the collection of all functions $f(\vec{x};) \in [\mathcal{F}; \mathcal{O}]$ which have only normal variables. Similarly, $(\text{NORMAL} \cap [\mathcal{F}; \mathcal{O}])_*$ denotes the collection of predicates whose characteristic function $f(\vec{x};)$ has only normal variables and belongs to $[\mathcal{F}; \mathcal{O}]$.

Define the following initial functions by

$$\begin{array}{rc}
\text{(0-ary constant)} & 0 \\
\text{(projections)} & I_j^{n,m}(x_1, \ldots, x_n; a_1, \ldots, a_m) = \begin{cases} x_j & \text{if } 1 \leq j \leq n \\ a_{j-n} & \text{if } n < j \leq n+m \end{cases} \\
\text{(successors)} & S_0(;a) = 2 \cdot a, \ S_1(;a) = 2 \cdot a + 1 \\
\text{(binary predecessor)} & P(;a) = \lfloor a/2 \rfloor \\
\text{(conditional)} & C(;a,b,c) = \begin{cases} b & \text{if } a \bmod 2 = 0 \\ c & \text{else.} \end{cases}
\end{array}$$

Definition 3.87 (Bellantoni-Cook [11]) The function f is defined by *safe composition* (SCOMP) from $g, u_1, \ldots, u_n, v_1, \ldots, v_m$ if

$$f(\vec{x}; \vec{a}) = g(u_1(\vec{x};), \ldots, u_n(\vec{x};); v_1(\vec{x}; \vec{a}), \ldots, v_m(\vec{x}; \vec{a})).$$

If $h(x;y)$ is defined, then SCOMP allows one to define

$$f(x,y;) = h(I_1^{2,0}(x,y;); I_2^{2,0}(x,y;)) = h(x;y).$$

However, one *cannot* similarly define $g(;x,y) = h(x;y)$.

Definition 3.88 The function f is defined by *safe recursion on notation*[22] (SRN) from the functions g, h_0, h_1 if

$$\begin{aligned}
&f(0,\vec{y};\vec{a}) = g(\vec{y};\vec{a})\\
&f(s_0(x),\vec{y};\vec{a}) = h_0(x,\vec{y};\vec{a}, f(x,\vec{y};\vec{a})), \text{ provided } x \neq 0\\
&f(s_1(x),\vec{y};\vec{a}) = h_1(x,\vec{y};\vec{a}, f(x,\vec{y};\vec{a})).
\end{aligned}$$

The function algebra B is defined by

$$[0, I, S_0, S_1, P, C; \text{SCOMP}, \text{SRN}].$$

Theorem 3.89 (Bellantoni-Cook [11]) *The polynomial time computable functions are exactly those functions of B having only normal arguments, i.e.*

$$\mathcal{F}\text{PTIME} = \text{NORMAL} \cap B.$$

The difficult direction of the proof is the inclusion from left to right. By Theorem 3.19 of Cobham, PTIME functions are those in the algebra

$$[0, I, s_0, s_1, \#; \text{COMP}, \text{BRN}].$$

To see the difficulties involved, suppose that f is defined by BRN from g, h_0, h_1 and that $g(\vec{y}) = g'(\vec{y};)$, $h_0(x,\vec{y},z) = h_0'(x,\vec{y},z;)$ and $h_1(x,\vec{y},z) = h_1'(x,\vec{y},z;)$. In trying to define f' by recursion on notation, one has $f'(0,\vec{y};) = g'(\vec{y};)$ and $f'(s_i(x),\vec{y};) = h_i'(x,\vec{y}, f'(x,\vec{y};);)$. However, this violates the requirement of SRN that the function value $f'(x,\vec{y};)$ be in a safe position in h_i'. For this reason a different approach is necessary.

Lemma 3.90 *If $f \in \mathcal{F}$PTIME then there exist $f' \in B$ and a monotone increasing polynomial p_f such that $f(\vec{x}) = f'(w;\vec{x})$ for all $|w| \geq p_f(|\vec{x}|)$.*

Proof. Temporarily, let's say that a function f is defined by polynomially bounded recursion on notation (PBRN) from g, h_0, h_1 if f is defined by recursion on notation from these functions, and additionally there exists a polynomial p satisfying $|f(x,\vec{y})| \leq p(|x|,|\vec{y}|)$ for all $x, \vec{y}$. Since $pad, \#$ are easily defined by PBRN [for instance, $0\#y = 1$, $s_i(x)\#y = pad(y, x\#y)$ where $|x\#y| \leq |x|\cdot|y|+1$] it follows from Theorem 3.19 that $\mathcal{F}\text{PTIME} = [0, I, s_0, s_1; \text{COMP}, \text{PBRN}]$. The lemma is now proved by induction on the construction of f in the latter algebra.

If f is $0, I_k^n, s_0, s_1$ then we may take f' to be the corresponding initial function of B and p_f to be 0. Suppose that $f(\vec{x}) = g(h_1(\vec{x}),\ldots,h_n(\vec{x}))$ is defined by composition, where by the induction hypothesis

$$g(y_1,\ldots,y_n) = g'(w;y_1,\ldots,y_n) \text{ for } |w| \geq p_g(|y_1|,\ldots,|y_n|) \tag{34}$$

$$h_i(\vec{x}) = h_i'(w;\vec{x}) \text{ for } |w| \geq p_{h_i}(|\vec{x}|). \tag{35}$$

Define

$$f'(w;\vec{x}) = g'(w; h_1'(w;\vec{x}),\ldots,h_n'(w;\vec{x})) \tag{36}$$

$$p_f(|\vec{x}|) = p_g(p_{h_1}(|\vec{x}|),\ldots,p_{h_n}(|\vec{x}|)) + \sum_{i=1}^{n} p_{h_i}(|\vec{x}|). \tag{37}$$

[22] In [11] this scheme is called *predicative notational recursion.*

It follows that $f(\vec{x}) = f'(w;\vec{x})$ for all $|w| \geq p_f(|\vec{x}|)$.

Suppose that f is defined from g, h_0, h_1 by PBRN as follows

$$f(0,\vec{y}) = g(\vec{y})$$
$$f(s_i(x),\vec{y}) = h_i(x,\vec{y},f(x,\vec{y}))$$

where $|f(x,\vec{y})| \leq q(|x|,|\vec{y}|)$ for some polynomial q. By the induction hypothesis then there exist $g', h_0', h_1', p_g, p_{h_0}, p_h$ satisfying

$$\begin{array}{rcl} g(\vec{y}) = g'(w;\vec{y}) & & \text{for } |w| \geq p_g(|\vec{y}|) \\ h_i(x,\vec{y},z) = h_i'(w;x,\vec{y},z) & & \text{for } |w| \geq p_{h_i}(|x|,|\vec{y}|,|z|). \end{array}$$

Let $E(z,w;x)$ be the initial segment of x obtained by removing from x the $|w| \dot{-} |z|$ lowest order bits. By SRN define F by

$$F(0,w;x,\vec{y}) = 0$$

$$F(s_i(z),w;x,\vec{y}) = \begin{cases} g'(w;\vec{y}) & \text{if Case 1} \\ h_0'(w;E(z,w;x),\vec{y},F(z,w;x,\vec{y})) & \text{if Case 2} \\ h_1'(w;E(z,w;x),\vec{y},F(z,w;x,\vec{y})) & \text{if Case 3} \\ F(z,w;x,\vec{y}) & \text{otherwise} \end{cases} \tag{38}$$

where

- Case 1 holds if $|w| - |x| = |s_i(z)| \leq |x|$,
- Case 2 holds if $|w| - |x| < |s_i(z)| \leq |x|$ and the low order bit of $E(s_i(z),w;x)$ is 0,
- Case 3 holds if $|w| - |x| < |s_i(z)| \leq |x|$ and the low order bit of $E(s_i(z),w;x)$ is 1.

To see that $F \in B$, introduce the following functions. The low order bit $M(;a) = a \bmod 2$ is defined by $M(;a) = C(;a,0,S_1(0))$. The truncation function $T(x;a) = \lfloor a/2^{|x|} \rfloor$ is defined by

$$\begin{array}{rcl} T(0;a) & = & a \\ T(s_i(x);a) & = & P(;T(x;a)). \end{array}$$

Let $T'(x,y;) = T(x,y;) = \lfloor y/2^{|x|} \rfloor$, and define the extraction operator

$$E(x,w;a) = T(T'(x,w;);a) = \lfloor a/2^{|w| \dot{-} |x|} \rfloor$$

so that $E(x,w;a)$ is the initial segment of a produced by removing from a the $|w| \dot{-} |x|$ lowest order bits. Define the bitwise OR function by

$$\begin{array}{rcl} \vee(0;a) & = & M(;a) \\ \vee(s_i(x);a) & = & C(;\vee(x;a),M(;T(s_i(x);a)),1). \end{array}$$

Note that for $|w| - |x| \leq |y| \leq |w|$ it follows that $|w| - |x| = |y|$ if and only if $\vee(w;E(y,w;x)) = 0$ and $|w| - |x| < |y|$ iff $\vee(w;E(y,w;x)) = 1$. It is now straightforward, to give a more formal definition of F placing it in B. Now set

$$f'(w;x,\vec{y}) = F(w,w;x,\vec{y}) \tag{39}$$

and

(40)

$$p_f(|x|,|\vec{y}|) = p_{h_0}(|x|,|\vec{y}|,q(|x|,|\vec{y}|)) + p_h(|x|,|\vec{y}|,q(|x|,|\vec{y}|)) + p_g(|\vec{y}|) + |x| + 1.$$

Claim 3.91 If u satisfies $|w|-|x| \leq |u| \leq |w|$ and $|w| \geq p_f(|x|, |\vec{y}|)$ then $F(u, w; x, \vec{y}) = f(E(u, w; x), \vec{y})$.

Proof of claim. Fix w satisfying $|w| \geq p_f(|x|, |\vec{y}|)$. Proceed by induction on $|u|$. First suppose that $|u| = |w| - |x|$. Then $E(u, w; x) = \lfloor x/2^{|w|-|u|} \rfloor = \lfloor x/2^{|x|} \rfloor = 0$. By (40), $|w| \geq |x| + 1$, so $|u| \geq 1$ and Case 1 applies. It then follows that

$$F(u, w; x, \vec{y}) = g'(w; \vec{y}) = f(E(u, w; x), \vec{y}).$$

Now suppose that $|w| - |x| < |u| \leq |w|$, and that $u = s_0(z)$ or $u = s_1(z)$. In the definition of $F(s_i(z), w; x, \vec{y})$ only case 2 or case 3 can occur.
Case 1 The $(|x| + |u| - |w|)$-th bit of x from the left is 0, or equivalently the $(|w| - |z| - 1)$-st bit of x from the right is 0. Then

$$\begin{aligned} F(s_i(z), w; x, \vec{y}) &= h_0'(w; E(z, w; x), \vec{y}, F(z, w; x, \vec{y})) \\ &= h_0'(w; E(z, w; x), \vec{y}, f(E(z, w; x), \vec{y})) && \text{induction hypothesis} \\ &= h_0(E(z, w; x), \vec{y}, f(E(z, w; x), \vec{y})) && \text{by justification below} \\ &= f(s_0(E(z, w; x)), \vec{y}) && \text{by definition of } f \text{, if } E(z, w; x) \neq 0 \\ &= f(E(s_i(z), w; x), \vec{y}) \end{aligned}$$

The last line follows, because in case 1, the low order bit of $E(s_i(z), w; x)$ is 0, so by the definition of E, $E(s_i(z), w; x) = s_0(E(z, w; x))$. The justification for the second line in the above equations is given as follows.

$$\begin{aligned} |w| &\geq p_{h_0}(|x|, |\vec{y}|, q(|x|, |\vec{y}|)) \\ &\geq p_{h_0}(|E(z, w; x)|, |\vec{y}|, q(|E(z, w; x)|, |\vec{y}|)) && \text{as } |E(z, w; x)| \leq |x| \text{ and } q \text{ is monotonic} \\ &\geq p_{h_0}(|E(z, w; x)|, |\vec{y}|, |f(E(z, w; x), \vec{y})|) && \text{as } q \text{ bounds length of } f \\ &\geq p_{h_0}(|E(z, w; x)|, |\vec{y}|, |F(z, w; x, \vec{y})|) && \text{induction hypothesis of claim.} \end{aligned}$$

Case 2 The $(|w| - |z| - 1)$-st bit of x from the left is 1.
This case in handled similarly to that of case 1, and so the proof of the claim is complete. ∎

By definition of E, it is clear that for all $|u| = |v|$ we have $E(u, w; x) = E(v, w; x)$. Using this, an easy induction on notation yields that for $|u| \geq |x|$ and $|w| \geq p_f(|x|, |\vec{y}|)$

$$F(u, w; x, \vec{y}) = F(x, w; x, \vec{y}).$$

For $|w| \geq p_f(|x|, |\vec{y}|)$ then

$$\begin{aligned} f'(w; x, \vec{y}) &= F(w, w; x, \vec{y}) && \text{by definition} \\ &= F(x, w; x, \vec{y}) && \text{as } |w| \geq |x| + 1 \\ &= f(E(x, w; x), \vec{y}) && \text{by Claim 3.91} \\ &= f(x, \vec{y}) && \text{by definition of } E. \end{aligned}$$

This completes the proof of the lemma. ∎

To show all functions of $\mathcal{F}$PTIME are functions of B containing only normal arguments, appropriate bounding functions in B must be defined.

Theorem 3.92 *If $f \in \mathcal{F}$PTIME then $f(\vec{x};) \in B$.*

Proof. Since f is polynomially bounded, let m, c be such that

$$|f(x_1, \ldots, x_n)| \leq (\sum_{i=1}^{n} |x_i|)^m + c.$$

Define

$$\begin{aligned} &Pad^2(0; y) = y \\ &Pad^2(s_i(x); y) = S_1(; Pad^2(x; y)) \\ &Pad^{k+1}(x_1, \ldots, x_k; x_{k+1}) = Pad^2(x_1; Pad^k(x_2, \ldots, x_k; x_{k+1})). \end{aligned}$$

Define

$$\begin{aligned} &Smash(0, x;) = 1 \\ &Smash(s_i(y), x;) = Pad^2(x; Smash(y, x;)) \end{aligned}$$

Then $Smash(y, x;) = 2^{|x| \cdot |y|}$. Let $b_0(x;)$ be obtained by composing $Smash(x, x;)$ with itself so as to satisfy $|b_0(x;)| \geq |x|^m + c$ and define

$$b(x_1, \ldots, x_n;) = b_0(Pad^{n+1}(x_1, \ldots, x_n; 1);).$$

Then $|f(x_1, \ldots, x_n)| \leq |b(x_1, \ldots, x_n;)|$ and so for f' given by Lemma 3.90, define F by $F(\vec{x};) = f'(b(\vec{x};); \vec{x})$. Then $F \in B$ and $f(\vec{x}) = F(\vec{x};)$.

∎

For the reverse inclusion, the following bounding lemma is proved by induction on the construction of f in B.

Lemma 3.93 *Let f belong to B. There is a monotone increasing polynomial q_f such that $|f(\vec{x}; \vec{y})| \leq q_f(|\vec{x}|) + \max_i |y_i|$ for all $\vec{x}, \vec{y}$.*

Theorem 3.94 (Bellantoni-Cook [11]) *If $f(\vec{x}; \vec{y}) \in B$ then there is $f'(\vec{x}, \vec{y}) \in \mathcal{F}$PTIME such that $f(\vec{x}; \vec{y}) = f'(\vec{x}, \vec{y})$ for all $\vec{x}, \vec{y}$.*

Proof. By induction on the construction of f in B. The case for initial functions and composition is straightforward. For monotonic bounding polynomial q_f given by the preceding lemma, there is a function $g \in [0, I, s_0, s_1, \#; \text{COMP}, \text{BRN}]$ satisfying $q_f(|\vec{x}|, |\vec{y}|) \leq |g(\vec{x}, \vec{y})|$. Thus SRN may be simulated using BRN. ∎

Corollary 3.95

$$\text{PTIME} = (\text{NORMAL} \cap [0, I, S_0, S_1, P, C; \text{SCOMP}, \text{SRN}])_*.$$

This approach has led to other characterizations of familiar complexity classes using *safe* variants of unbounded recursion schemes.

Theorem 3.96 (Bellantoni [9]) *Let $f(\vec{x})$ be a function satisfying $|f(\vec{x})| = O(\log |x|)$. Then $f(\vec{x})$ is computable by a logspace Turing machine iff*

$$f(\vec{x};) \in [0, I, S_1, P, C; \text{SCOMP}, \text{SRN}].$$

Note that the function S_0 does not belong to the above algebra, so the intuition is that for *small* functions (of logarithmic growth rate), LOGSPACE computations are arithmetized by using unary numerals on the work tape along with the same closure operators as for polynomial time. Bellantoni first proves his result for the operation of *simultaneous* safe recursion on notation, and then simulates this simultaneous scheme by SRN, a non-trivial task since the usual pairing function uses S_0. The previous theorem yields a nice characterization of LOGSPACE, to be compared with Corollary 3.95.

Corollary 3.97

$$\text{LOGSPACE} = (\text{NORMAL} \cap [0, I, S_1, P, C; \text{SCOMP}, \text{SRN}])_*.$$

Definition 3.98 The function f is defined by *safe minimization* (SMIN) from the function g, denoted $f(\vec{x};\vec{b}) = s_1(\mu a[g(\vec{x};a,\vec{b}) \bmod 2 = 0)])$, if

$$f(\vec{x};\vec{b}) = \begin{cases} s_1(\min\{a : g(\vec{x};a,\vec{b}) = 0\}), & \text{if such exists,} \\ 0 & \text{else.} \end{cases}$$

The algebra $\mu B = [0, I, S_0, S_1, P, C; \text{SCOMP}, \text{SRN}, \text{SMIN}]$. Let μB_i denote the set of functions derivable in μB using at most i applications of safe minimization.

Theorem 3.99 (Bellantoni [10])

$$\square_i^P = \{f(\vec{x};) : f \in \mu B_i\}.$$

Definition 3.100 (Bellantoni [9]) The function f is defined by *safe recursion*[23] (SR) from the functions g, h if

$$f(0, \vec{y}; \vec{a}) = g(\vec{y}; \vec{a})$$
$$f(x+1, \vec{y}; \vec{a}) = h(x, \vec{y}; \vec{a}, f(x, \vec{y}; \vec{a})).$$

Define the following initial functions by

(41) (successor) $S(;a) = a + 1$

(42) (predecessor) $Pr(;a) = a \dot{-} 1$

(43) (conditional) $K(;a,b,c) = \begin{cases} b & \text{if } a = 0 \\ c & \text{else.} \end{cases}$

Recall that $\mathcal{E}^2$, the second level of the Grzegorczyk hierarchy, is the collection of linear space computable functions.

Theorem 3.101 (Bellantoni [9])

$$\mathcal{E}^2 = \text{NORMAL} \cap [0, I, S, Pr, K; \text{SCOMP}, \text{SR}].$$

W. Handley (unpublished) and D. Leivant (unpublished) both independently obtained Theorem 3.101. Building on Bellantoni's proof, in her work on linear space reasoning, A.P. Nguyen [103] gave a slightly different characterization of this class. In [33] the author gave a *safe* characterization of ETIME functions of linear growth, by adapting the proof of Theorem 3.84.

In [91] D. Leivant gave an alternative formulation of the safe characterizations of polynomial time and of linear space, by introducing a *tiering* notion to arbitrary word algebras. The idea is that one admits various copies or *tiers* W_0, W_1, ... of the word algebra W (generated from 0 by s_0, s_1),[24] and defines *ramified recurrence* by

$$f(s_i(x), \vec{y}) = h_i(f(x, \vec{y}), x, \vec{y})$$

[23] In [9] this scheme is called *predicative primitive recursion*.

[24] Leivant considers more general algebras defined from finitely many constructors.

where the tier of the first argument $s_i(x)$ is larger than the tier of the value $f(x,\vec{y})$. Comparing with the Bellantoni-Cook notation, tier 0 is safe, whereas tier 1 is normal. Leivant then shows that f is computable by a register machine over algebra A in time polynomial in the length of the inputs iff f is definable by explicit definition (corresponding essentially to safe composition) and ramified recurrence over A. This yields that f is polynomial time computable iff f is definable by explicit definition and ramified recurrence over W_0, W_1, whereas f is linear space computable (i.e. in $\mathcal{E}^2$) iff f is definable by explicit definition and ramified recurrence over $\mathbf{N}$, the unary algebra defined from 0, S.

These characterizations of complexity classes in terms of safe operations suggests the following problem.

Problem 3.102 Characterize the classes $\mathcal{M}^n$, for $n = 0, 1, 2$ and for each $n \geq 0$, the Grzegorczyk class $\mathcal{E}^n$ via appropriate initial functions, and safe operations. In particular, can one characterize $\mathcal{M}^2$ by $[0, I, S, Pr, K; \text{SCOMP}, \text{SMIN}]$? (Note that the conditional function $cond \in \mathcal{E}^1 - \mathcal{E}^0$.)

Turning to parallel computation, by building on Theorem 3.27, S. Bellantoni [9] characterizes NC as those functions with normal variables in an algebra built up from 0, I, S_0, S_1, the conditional C, the bit function BIT, the length function $L(;a) = |a|$, a variant $\#'$ of the smash function, and closed under safe composition, concatenation recursion on notation and a *safe* version of weak bounded recursion on notation. Define the *half* function by $H(x) = \lfloor x/(2^{\lceil |x|/2 \rceil}) \rfloor$, and note that the least number of times which H can be iterated on x before reaching 0 is $||x||$. The function f is defined by ***safe weak recursion on notation*** (SWRN) [25] from the functions g, h if

$$
\begin{aligned}
f(0,\vec{y};\vec{a}) &= g(\vec{y};\vec{a}) \\
f(x,\vec{y};\vec{a}) &= h(x,\vec{y};\vec{a}, f(H(x),\vec{y};\vec{a})), \text{provided } x \neq 0.
\end{aligned}
$$

Theorem 3.103 (S. Bellantoni [9])

$$\text{NC} = [0, I, S_0, S_1, C, L, \text{BIT}, \#'; \text{SCOMP}, \text{CRN}, \text{SWRN}].$$

Following [3], define $\text{BH}(x) = x \bmod 2^{\lceil |x|/2 \rceil}$ and $\text{FH}(x) = msp(x, \text{BH}(x))$. The *back half* $\text{BH}(x)$ consists of the $\lceil |x|/2 \rceil$ rightmost bits of x, while the *front half* $\text{FH}(x)$ consists of the $\lfloor |x|/2 \rfloor$ leftmost bits of x. In [14] S. Bloch defines two distinct safe versions of Allen's divide and conquer recursion.

Definition 3.104 (S. Bloch [14]) The function f is defined by ***safe divide and conquer recursion*** (SDCR) from the functions g, h if

$$
f(x,y,\vec{z};\vec{a}) = \begin{cases} g(x,\vec{z};\vec{a}) & \text{if } |x| \leq \max(|y|,1) \\ h(x,y,\vec{z};\vec{a}, f(\text{FH}(;x),y,\vec{z};\vec{a}), f(\text{BH}(;x),y,\vec{z};\vec{a})) & \text{else.} \end{cases}
$$

The function f is defined by ***very safe divide and conquer recursion*** (VSDCR) from the functions g, h if

$$
f(x,y,\vec{z};\vec{a}) = \begin{cases} g(x,\vec{z};\vec{a}) & \text{if } |x| \leq \max(|y|,1) \\ h(;x,\vec{z},\vec{a}, f(\text{FH}(;x),y,\vec{z};\vec{a}), f(\text{BH}(;x),y,\vec{z};\vec{a})) & \text{else.} \end{cases}
$$

Note that in VSDCR the iteration function h has no normal parameters, and hence cannot itself be defined by recursion.

[25] In [9] this scheme is called *log recursion.*

Theorem 3.105 (S. Bloch [14]) *There is a collection* BASE *of initial functions, for which*

$$\text{ALOGTIME} = (\text{NORMAL} \cap [\text{BASE}; \text{SCOMP}, \text{VSDCR}])_*$$

$$\text{POLYLOGTIME} = (\text{NORMAL} \cap [\text{BASE}; \text{SCOMP}, \text{SDCR}])_*.$$

Sketch of Proof. The collection BASE of initial functions consists of NC^0 computable versions of MSP, LSP, FH, BH, a conditional function, and some string manipulating functions (see [14] for details).

Only the proof of the first assertion will be sketched. Consider first the inclusion $[\text{BASE}; \text{SCOMP}, \text{VSDCR}]_* \subseteq \text{ALOGTIME}$. Show that $\text{BASE} \subseteq \text{NC}^0 \subseteq \text{ALOGTIME}$. Since the iterating function $h(; z, \vec{x}, \vec{y}, u, v)$ has only safe parameters, h must be obtained from BASE by safe composition, hence belongs to NC^0. Very safe divide and conquer recursion corresponds to the evaluation of a binary tree of logarithmic depth, whose leaves correspond to $g(x, \vec{z}; \vec{a})$ for $|x| \leq \max(|y|, 1)$, and whose internal nodes correspond to the NC^0 function h. Since the resulting circuit is of logarithmic depth, it follows that the function f defined by VSDCR belongs to ALOGTIME.

Now consider the inclusion $\text{ALOGTIME} \subseteq [\text{BASE}; \text{SCOMP}, \text{VSDCR}]_*$. Without using VSDCR, define certain string manipulating functions explicitly. Let M be a RATM. The computation tree of M corresponds to a binary branching logdepth tree, all nodes of which are encodings of the current work tape, index tapes, state and tape head positions. Without loss of generality, one may assume that a bit of the input can be queried, using the index tape, only at a leaf configuration. Depending on the current contents (say i) of the index tape, a bit (say the i-th bit) of the input is accessed. Depending on that query, evaluation of the leaves of the computation tree is defined, and evaluation of the internal nodes involves the simple evaluation of an AND-OR tree (minimax strategy). Describe the leaf nodes by a function in [BASE; SCOMP, VSDCR]. Evaluation of the AND-OR tree is very simply described, using an iterating function having only safe parameters. ∎

It seems clear that linear time on multitape Turing machines or on random access machines can be characterized using appropriate initial functions, closure under safe composition and some form of simultaneous very safe recursion (simultaneous recursion, since a pairing function apparently cannot be defined from the initial functions using safe composition – such would be necessary for defining NEXT_M). Details have been worked out by S. Bloch [15] and J. Otto [106, 107, 108], the latter using category theory.

4 Type 2 functionals

Many programming languages allow functions to be passed as parameters to other functions or procedures. For instance, in different limited manners PASCAL and C allow function parameters, while C^{++} supports function templates and ADA, ML admit polymorphism.[26] The oracle Turing machine is a reasonable construct to model function parameter passing, though it has principally been used to study reducibilities $A \leq_T B$, $A \leq_T^P B$ etc. between *sets*. Nevertheless, higher type *functional* complexity theory is a new area with fundamental open problems. In particular, though various classes have been proposed as candidates for the *feasible* type 2 functionals, there is not yet general agreement about the right notion. For

[26]Polymorphism allows function and procedures to abstract over data types — e.g. a generic sorting algorithm for any data type having a comparison function.

reasons of space, only a few recent directions in higher type functional complexity will be presented. For more information, see the survey [42] by S.A. Cook and the volume edited by D. Leivant [92] (higher type complexity theory) and the articles by D. Norman and H. Schwichtenberg in this volume (higher type recursion theory).

Definition 4.1 A *type 2* functional F of *rank* (k, ℓ) is a total mapping from $(\mathbf{N}^N)^k \times \mathbf{N}^\ell$ into $\mathbf{N}$.

It is worth noting[27] that at type 2, Gödel's question about classification of recursive functions is completely answered. Namely, a rank $(1,1)$ recursive functional F has normal form

$$F(f,x) = U(\mu y[T_F(x, \overline{f}(y)) = 0])$$

where $\overline{f}(y) = \langle f(0), \ldots, f(y-1) \rangle$ and T_F is a well founded tree of height $< \omega_1^{ck}$. Thus type 2 total recursive functionals can be classified with respect to recursive ordinals.

Definition 4.2 A function oracle Turing machine (OTM) is a Turing machine M which in addition to read-only input tape, distinguished output tape and finitely many work tapes, has an *oracle query tape* and *oracle answer tape*, both one-way infinite, for each *function* input. Additionally M has a special oracle query state for each function input.

Note that the previous definition, unlike Definition 2.5, allows function (rather than set) arguments. In order to query a function input f at x, the machine M takes steps to write x in binary on the oracle query tape. When the oracle query tape head is in its leftmost square, M enters a special query state. In the next step, M erases both the oracle query and answer tapes, writes the function value $f(x)$ in binary on the oracle answer tape, and leaves the oracle query and answer tape heads in their leftmost squares. Upon entering the oracle query state, there seem to be two natural measures for the time to complete the function query $f(x)$. The *unit cost*, considered by Mehlhorn [96], charges unit time, while the *function length cost*, considered by Constable [41] and later Kapron and Cook [80], charges $\max\{1, |f(x)|\}$ time. The machine M computes the $rank(n, m)$ functional $F(f_1, \ldots, f_n, x_1, \ldots, x_m)$ if M has n oracle query states, query and answer tapes corresponding to $f_1, \ldots, f_n$ and if M outputs the integer $F(f_1, \ldots, f_n, x_1, \ldots, x_m)$ in binary on the output tape, when started in its initial state q_0 with input tape $\underline{B}x_1 B x_2 B \cdots B x_m B$.

Definition 4.3 For any OTM M, for any inputs $f_1, \ldots, f_n, x_1, \ldots, x_m$ and integer t, the query answer set $QA_M(\vec{f}, \vec{x}, t)$ is defined as

$$\{(y,z) : M \text{ on input } \vec{f}, \vec{x} \text{ queries some } f_i(y) = z \text{ within time } S(t) \text{ steps}\}$$

where $S(t)$ is the least number of steps s for which if M runs s steps then its time complexity is at least t. The *query set* $Q_M(\vec{f}, \vec{x}, t)$ is $\{y : (\exists z)[(y,z) \in QA_M(\vec{f}, \vec{x}, t)]\}$ and the *answer set* $A_M(\vec{f}, \vec{x}, t)$ is $\{z : (\exists y)[(y,z) \in QA_M(\vec{f}, \vec{x}, t)]\}$.

An OTM M is a *polynomial time oracle Turing machine* (POTM) if M computes a total $rank(n, m)$ functional F and there is a polynomial p such that for all input $f_1, \ldots, f_n, x_1, \ldots, x_m$ and times t

$$t \leq p(|\max(\{x_1, \ldots, x_m\} \cup A_M(\vec{f}, \vec{x}, t))|).$$

OPT is the collection of type 2 functionals computable by an oracle polynomial time oracle Turing machine.

[27] This well-known fact, pointed out to the author by S.S. Wainer, is proved in H. Schwichtenberg's article in this volume.

Example 4.4

(1) $F(f,x) = \max\{f(y) : y \leq |x|\}$ belongs to OPT.

(2) $G(f,x) = \max\{f(y) : |y| \leq |x|\}$ does not belong to OPT.

(3) $H(f,x) = f^{(|x|)}(x)$ belongs to OPT.

In [96] K. Mehlhorn extended Cobham's function algebra to type 2 functionals. A modern presentation of Mehlhorn's definition uses the following schemes.

Definition 4.5 (Townsend [136]) F is defined from $H, G_1, \ldots, G_m$ by *functional composition* if for all $\vec{f}, \vec{x}$,

$$F(\vec{f},\vec{x}) = H(\vec{f}, G_1(\vec{f},\vec{x}), \ldots, G_m(\vec{f},\vec{x}), \vec{x}).$$

F is defined from G by *expansion* if for all $\vec{f}, \vec{g}, \vec{x}, \vec{y}$,

$$F(\vec{f},\vec{g},\vec{x},\vec{y}) = G(\vec{f},\vec{x}).$$

F is defined from $G, G_1, \ldots, G_m$ by *functional substitution* if for all $\vec{f}, \vec{x}$,

$$F(\vec{f},\vec{x}) = H(\vec{f}, \lambda y.G_1(\vec{f},\vec{x},y), \ldots, \lambda y.G_m(\vec{f},\vec{x},y), \vec{x}).$$

F is defined from G, H, K by *limited recursion on notation*[28] (LRN) if for all $\vec{f}, \vec{x}, y$,

$$\begin{aligned} F(\vec{f},\vec{x},0) &= G(\vec{f},\vec{x}) \\ F(\vec{f},\vec{x},y) &= H(\vec{f},\vec{x},y,F(\vec{f},\vec{x},\lfloor \tfrac{y}{2} \rfloor)), \text{ if } y \neq 0 \end{aligned}$$

provided that $F(\vec{f},\vec{x},y) < K(\vec{f},\vec{x},y)$ holds for all $\vec{f},\vec{x},y$.

Definition 4.6 (Townsend [136], Kapron, Cook [44]) Let X be a class of type 2 functionals. The class of *basic feasible functionals* defined from X, denoted BFF(X), is the smallest class of functionals containing X, 0,s_0,s_1,I,$\#$ and the application functional Ap, defined by $Ap(f,x) = f(x)$, and which is closed under functional composition, expansion, and LRN. If $F \in$ BFF(X), then F is basic feasible in X. The class BFF of basic feasible functionals[29] is BFF$(\emptyset)$.

In [96] Mehlhorn introduced the Turing machine model with function oracle, charging unit cost for a function oracle call, independent of the length of the function value returned. Mehlhorn's model has an oracle input tape and an oracle output tape, thus avoiding the situation where m successive iterates of a function $f(f(\ldots f(x)\ldots))$ might take m steps. Using the techniques of low-level arithmetization from the proof of Theorem 3.19, the following result is proved.

Theorem 4.7 (Mehlhorn [96]) *For every functional F in* BFF*, there is a unit cost model* OTM *M which computes F, i.e. $M(\vec{f},\vec{x}) = F(\vec{f},\vec{x})$, and where the runtime of M on all input $\vec{f},\vec{x}$ is bounded by $|G(\vec{f},\vec{x})|$ for some G belonging to* BFF*. Conversely, if functional F is computed by* OTM *M, which on input $\vec{f},\vec{x}$ has runtime at most $|G(\vec{f},\vec{x})|$ for some G belonging to* BFF*, then $F \in$* BFF.

[28] This scheme is clearly equivalent to that of bounded recursion on notation BRN for functionals, yet notationally easier to manipulate in the proofs which follow.

[29] Townsend [136] calls this class **POLY**. Cook and Kapron call this class *basic feasible*, leaving open the possibility that with future research a more natural class of *feasible* functionals may be investigated. The original definition of basic feasible functional required closure under functional substitution, but this can be defined from the remaining schemes, as noted in [136].

Definition 4.8 A class $\mathcal{F}$ of type 2 functionals has the *Ritchie-Cobham property* if

$$\mathcal{F} = \{F : \text{there exist } G \in \mathcal{F} \text{ and OTM } M \text{ which on any input } \vec{f}, \vec{x} \text{ computes } F(\vec{f}, \vec{x}) \text{ within time } |G(\vec{f}, \vec{x})|\}.$$

With this definition, Theorem 4.7 can be rephrased by the statement that BFF has the Ritchie-Cobham property using unit cost OTM.

It is clear that OPT contains functionals which are not intuitively feasible. In particular, substituting the polytime computable function $\lambda y.y^2$ for f in H, where $H(f,x) = f^{(|x|)}(x)$, above yields $H(\lambda y.y^2, x) = x^{2^{|x|}}$ which is not a polytime computable type 1 function (example due to A. Seth [125]). The following example, due to S. Cook, provides a functional which belongs to OPT yet not to BFF.

Let $\preceq$ quasi-order $\mathbf{N} \times \mathbf{N}$ by *length first difference*; i.e. $(a,b) \preceq (c,d)$ iff $|a| < |c|$ or ($|a| = |c|$ and $|b| \leq |d|$). Transfer this ordering to $\mathbf{N}$ by a standard polynomial time pairing function. Define the $rank(1,0)$ functional L by $L(f) = \mu i[(\exists j < i)(f(j) \preceq f(i))]$. Note that $\preceq$ defines a well quasi-ordering on $\mathbf{N} \times \mathbf{N}$, so L is well defined.

Theorem 4.9 (S. Cook [42]) *The functional L belongs to* OPT *yet not to* BFF.

S. Cook [42] points out that the type-1 section of the closure of OPT with L is just the type-1 section of OPT, i.e. the class of polynomial time computable functions, and so L should be considered a feasible functional. This argument suggests that BFF should not be considered the class of all feasible type-2 functionals. Against this, in [125] A. Seth proves that the type-1 section of the closure of type-2 exponential time with L is not the class of exponential time computable functions, and hence L should not be considered a feasible functional. It is worth noting that Bellantoni (p. 85 in [9]) showed that if one adds the length function $|x|$ to a modification of class B from [9], and closes under lambda abstraction and application, then the resulting higher type class *does* compute the functional L. S. Bellantoni (private correspondence) has raised the question whether the class obtained by omitting $|x|$ is equivalent to BFF.

In [80], Kapron and Cook lift Cobham's characterization of polynomial time computable functions to functionals of level 2. To state their result, the notion of *length* of a function and that of second order polynomial must be introduced.

Definition 4.10 The length $|f|$ of one-place function f is itself a one-place function defined by

$$|f|(n) = \max_{|x| \leq n} \{|f(x)|\}.$$

Let $f_1, \ldots, f_m$ be variables ranging over $\mathbf{N}^N$ and $x_1, \ldots, x_n$ be variables ranging over $\mathbf{N}$. The collection C of second order polynomials $P(f_1, \ldots, f_m, x_1, \ldots, x_n)$ is defined inductively as follows.

- *(i)* for any integer c, $c \in C$,
- *(ii)* for every $1 \leq i \leq n$, $x_i \in C$,
- *(iii)* if $P, Q \in C$ then $P + Q \in C$ and $P \cdot Q \in C$,
- *(iv)* if $P \in C$ then $f_i(P) \in C$ for $1 \leq i \leq m$.

The depth $d(P)$ of a second order polynomial P is defined inductively by $d(c) = 0 = d(x_i)$, $d(P+Q) = d(P \cdot Q) = \max(d(P), d(Q))$, $d(f_i(P)) = 1 + d(P)$. For any f and $Q \subseteq \mathbf{N}$, let f_Q be defined by

$$f_Q(x) = \begin{cases} f(x) & \text{if } x \in Q \\ 0 & \text{else.} \end{cases}$$

If $Q = Q_M(f,x,t)$ then M on inputs f_Q, x behaves identically to M on inputs f, x.

Fact 4.11 Suppose that M is an OTM and P a second order polynomial such that the runtime of M on inputs f, x is bounded by $P(|f|, |x|)$.

(i) Suppose that $Q = Q_M(f, x, t)$ and $t \geq P(|f_Q|, |x|)$. Then M halts within t steps.
(ii) Suppose that $Q = Q_M(f, x, t)$ and $Q' = Q_M(f, x, P(|f_Q|, |x|))$. Then either M halts within $P(|f_Q|, |x|)$ steps or $Q \subset Q'$.

The preceding fact is clear, since in *(i)* M on inputs f, x makes identical moves as M on f_Q, x, and in *(ii)* if $Q = Q'$, then apply (i) with Q' in place of Q.

Theorem 4.12 (B. Kapron and S. Cook [80]) BFF *is the collection of functionals $F(f_1, \ldots, f_n, x_1, \ldots, x_m)$ computable in time $P(|f_1|, \ldots, |f_n|, |x_1|, \ldots, |x_m|)$ for some second order polynomial P on an* OTM *with function length cost.*[30]

Proof. The inclusion from left to right is straightforward. Consider the inclusion from right to left. Suppose that M computes a functional F of rank $(1,1)$ and the runtime of M is bounded by the depth d second order polynomial P.

For $1 \leq c \leq d$, let $P^c_1, \ldots, P^c_{k_c}$ be an enumeration of depth c subpolynomials of P of the form $f(Q)$, where Q is of depth $c-1$. If $P^c_i = f(Q)$ then denote the associated Q by Q^c_i. Note that $d(P^c_i) = c$ and $d(Q^c_i) = c - 1$. For any Q^c_i there is a first order polynomial q^c_i satisfying

$$q^c_i(P^1_1(\vec{f}, \vec{x}), \ldots, P^1_{k_1}(\vec{f}, \vec{x}), \ldots, P^{c-1}_1(\vec{f}, \vec{x}), \ldots, P^{c-1}_{k_{c-1}}(\vec{f}, \vec{x}), \vec{x}) = Q^c_i(\vec{f}, \vec{x}).$$

For any inputs f, x of M and time t, there exist queries $q_1, \ldots, q_d$ in $Q = Q_M(f, x, t)$ such that for $1 \leq c \leq d$

$$|q_c| \leq \max\{Q^c_1(|f_Q|, |x|), \ldots, Q^c_{k_c}(|f_Q|, |x|)\} \tag{44}$$

and

$$|f(q_c)| \geq \max\{P^c_1(|f_Q|, |x|), \ldots, P^c_{k_c}(|f_Q|, |x|)\}. \tag{45}$$

For $1 \leq c \leq d$ there exist first order polynomials q_c satisfying

$$Q^c_i(|f_Q|, |x|) \leq q_c(|f(q_1)|, \ldots, |f(q_{c-1})|, |x|).$$

As well there is a first order polynomial p such that

$$P(|f_Q|, |x|) \leq p(|f(q_1)|, \ldots, |f(q_d)|, |x|).$$

For every first order polynomial q, there exists a function h_q built up from $0, x_i, s_0, s_1, \#$ using composition, such that

$$q(|x_1|, \ldots, |x_n|) \leq |h_q(x_1, \ldots, x_n)|.$$

For example, $|x|^2 \cdot (|y| + 3)$ is bounded by $|(x\#x)\#(s_1(s_1(s_1(y))))|$. It follows that there exist BFF functionals $\overline{Q_c}$, $1 \leq c \leq d$, and $\overline{P}$ such that

$$Q^c_i(|f_Q|, |x|) \leq |\overline{Q_c}(f, q_1, \ldots, q_{c-1}, x)|$$

and

$$P(|f_Q|, |x|) \leq |\overline{P}(f, q_1, \ldots, q_d, x)|.$$

For $1 \leq c \leq d$ define maxquery^c_M of rank (1,2) by

$$\text{maxquery}^c_M(f, x, r) = \mu q_c \in Q(M, f, x, t)[q_c \text{ satisfies (44) and (45)}],$$

[30] As noted in [34], this result holds as well for unit cost.

where t is the least time satisfying $|Q(M,f,x,t)| \geq r$ or M halts in t steps. Define A_{lmax} to be the rank (1,1) functional satisfying

$$|f(A_{lmax}(f,x))| = \max_{y \leq |x|}\{|f(y)|\}.$$

Note that A_{lmax} is BFF, since it can be defined by LRN as follows

$$\begin{array}{l} A_{lmax}(f,0) = 0 \\ A_{lmax}(f,x) = \begin{cases} A_{lmax}(f, \lfloor \frac{x}{2} \rfloor) & \text{if } |f(|x|)| \leq |f(A_{lmax}(\lfloor \frac{x}{2} \rfloor))| \\ |x| & \text{otherwise} \end{cases} \end{array}$$

and $A_{lmax}(f,x) \leq |x|$ for all f, x.

Assuming that for $1 \leq c \leq d$, maxquery^c_M is BFF, we can show that F is BFF. Let

$$\begin{array}{lr} r_1 = \overline{Q_1}(f,x) & \\ T_c = \overline{P}(f, \text{maxquery}^1_{M(f,x,r_c)}, \ldots, \text{maxquery}^d_{M(f,x,r_c)}, x) & \text{for } 1 \leq c \leq d \\ l_c = A_{lmax}(f, 2\#T_c) & \text{for } 1 \leq c \leq d \\ r_c = \overline{Q_c}(f, l_1, \ldots, l_{c-1}, x) & \text{for } 2 \leq c \leq d. \end{array}$$

Finally define G by

$$G(f,x) = \max\{\overline{P}(f, l_1, \ldots, l_d, x), \max_{1 \leq c \leq d} T_c\}.$$

Claim 4.13 M halts within $|G(f,x)|$ steps on inputs f, x.

Proof of claim. Suppose that $q_1, \ldots, q_d$ are as in (44) and (45). Then $|q_1| \leq |r_1|$ and

$$T_1 = \overline{P}(f, \text{maxquery}^1_{M(f,x,r_1)}, \ldots, \text{maxquery}^d_{M(f,x,r_1)}, x).$$

If M halts in $|T_1|$ steps, then surely M halts in $|G(f,x)|$ steps. If not, then M must have made more than r_1 queries to the oracle f. Thus

$$l_1 = A_{lmax}(f, 2\#T_1)$$

so

$$\begin{aligned} |f(l_1)| &= f(A_{lmax}(f, 2\#T_1)) \\ &= \max\nolimits_{y \leq |2\#T_1|} |f(y)| \\ &= \max\nolimits_{y \leq 2 \cdot |T_1|+1} |f(y)| \\ &\geq \max\nolimits_{y \leq 2\, r_1+1} |f(y)| \\ &\geq \max\nolimits_{|y| \leq |r_1|} |f(y)| \\ &= |f|(|r_1|). \end{aligned}$$

The only non-obvious step above relies on the observation that $|T_1| \geq r_1$, since when executing M for $|T_1|$ steps, either M halts or M asks more than r_1 oracle queries in that time. As M is assumed not to halt in $|T_1|$ steps, the r_1 oracle queries were posed in $|T_1|$ steps, so $r_1 \leq |T_1|$. Since $|q_1| \leq |r_1|$, it follows that $|f(q_1)| \leq |f|(|r_1|) \leq |f(l_1)|$. Now

$$r_2 = \overline{Q_2}(f, l_1, x)$$

and

$$T_2 = \overline{P}(f_1, \text{maxquery}^1_M(f,x,r_2), \ldots, \text{maxquery}^d_M(f,x,r_2), x).$$

If M halts in $|T_2|$ steps, then M halts in $|G(f,x)|$ steps. Otherwise, $l_2 = A_{lmax}(f, 2\#T_2)$ and a similar argument as before yields

$$|f(l_2)| \geq |f|(|r_2|).$$

As well,

$$\begin{aligned} |q_2| &\leq \max_{1 \leq i \leq k_2} Q_i^2(|f|,|x|) \\ &\leq |\overline{Q_2}(f,q_1,x)| \\ &\leq |r_2|. \end{aligned}$$

Hence $|f(q_2)| \leq |f|(|r_2|) \leq |f(l_2)|$. Proceeding inductively, if M does not halt in $|T_i|$ steps for some $1 \leq i \leq d$, then it is the case that

$$|q_i| \leq |r_i|, \quad r_i \leq |T_i|, \text{ and } |f(q_i)| \leq |f|(|r_i|) \leq |f(l_i)|$$

so that M halts in

$$\begin{aligned} P(|f|,|x|) &\leq p(|f(q_1)|,\ldots,|f(q_d)|,|x|) \\ &\leq |\overline{P}(f,q_1,\ldots,q_d,x)| \end{aligned}$$

steps. Thus M halts in $|G(f,x)|$ steps. This establishes the claim. ■

Since a BFF functional $Run_M(f,x,T)$ can be defined which arithmetizes the execution of OTM M on inputs f,x for $|T|$ steps, it follows that $F(f,x) = Output_M(Run_M(f,x,G(f,x)))$, where $Output_M$ is an easily defined BFF functional. To establish that F is BFF, it only remains to prove that $\text{maxquery}_M^c(f,x,r)$ is BFF for $1 \leq c \leq d$. But this follows from the type 2 version of Theorem 3.86. This completes the proof of Theorem 4.12.

■

The *oracle concurrent random access machine* (OCRAM), introduced by the author, A. Ignjatovic and B. Kapron in [34] has instructions for *(i)* local operations — addition, cutoff subtraction, shift, *(ii)* global and local indirect reading and writing, *(iii)* control instructions — GOTO, conditional GOTO and HALT, *(iv)* oracle calls, where in one step, all active processors simultaneously can retrieve

$$f(x_i \cdots x_j) = f(\sum_{k=i}^{j} x_k \cdot 2^{j-k})$$

where i,j are current values of local registers, and x_i is the 0,1 value held in the i-th oracle register. If M on argument f,x runs in time $T(|f|,|x|)$ with $P(|f|,|x|)$ processors, then the formal details of the model ensure that $|u| \leq T(|f|,|x|) \cdot P(|f|,|x|)$ for every oracle call $f(u)$. If T,P are bounded by second order polynomials, then it follows that there is a second order polynomial Q, such that $|f(u)| \leq Q(|f|,|x|)$ for all oracle calls $f(u)$ during the computation of M on input f,x.

The OCRAM is formally defined as follows. For each k-ary function argument f, there are k infinite collections of *oracle registers*, the i-th collection labeled $M_0^{o,i}, M_1^{o,i}, M_2^{o,i}, \ldots$, for $1 \leq i \leq k$. As with global memory, in the event of a write conflict the lowest numbered processor succeeds in writing to an oracle register. Let res (result), $op0$ (operand 0) and $op1$ (operand 1) be non-negative integers, as well as $op2$,$op3$,…,$op(2k)$.

In addition to the instructions for the CRAM, the OCRAM has instructions concerning the oracle registers and oracle calls.

$*M^o_{res} \ := \ 0$

$$
\begin{array}{rcl}
*M^o_{res} & := & 1 \\
M^o_{res} & := & 0 \\
M^o_{res} & := & 1 \\
M_{res} & := & *M^o_{op1} \\
M_{res} & := & f([M_{op1} \cdots M_{op2}]_1, [M_{op3} \cdots M_{op4}]_2, \ldots, [M_{op(2k-1)} \cdots M_{op(2k)}]_k)
\end{array}
$$

The notation $[M_{op(2i-1)} \cdots M_{op(2i)}]_i$ denotes the integer whose binary notation is given in oracle registers $M^{o,i}_{M_{op(2i-1)}}$ through $M^{o,i}_{M_{op(2i)}}$. In other words,

$$[M_{op(2i-1)} \cdots M_{op(2i)}]_i = \sum_{m=M_{op(2i-1)}}^{M_{op(2i)}} M^{o,i}_m \cdot 2^{op(2i)-m}.$$

The instruction $*M^o_{res} := 0$ sets the contents of the oracle register whose address is given by the current contents of local memory M_{res} to 0. Similarly for the instruction $*M^o_{res} := 1$. The instruction $M_{res} := *M^o_{op1}$ sets the contents of local memory M_{res} to be the current contents of the oracle register whose address is given by the current contents of local memory M_{op1}. With these instructions, it will be the case that oracle registers hold a 0 or 1 but no larger integer. If any register occurring on the right side of an instruction contains 'B' meaning undefined, then the register on the left side of the instruction will be assigned the value 'B' (undefined). For instance, if a unary oracle function f is called in the instruction

$$M_{res} := f([M_{op1} \cdots M_{op2}])$$

and if some register M_i contains 'B', where $op1 \leq i \leq op2$, then M_{res} is assigned the value 'B'.

In characterizing AC^k in the non-oracle case, Stockmeyer and Vishkin [133] require a polynomial bound $p(n)$ on the number of active processors on inputs of length n. With the above definition of OCRAM one might hope to characterize the class of type 2 functionals computable in constant parallel time with a second-order polynomial number of processors as exactly the type 2 functionals in the algebra $\mathcal{A}_0$. Using the definitions given so far, this is not true. To rectify this situation, proceed as follows.

Definition 4.14 For every OCRAM M, functions f, g and integers x, t the query set $Q(M, f, x, t, g)$ is defined as

$$\{y : \ M \text{ with inputs } f, x \text{ queries } f \text{ at } y \text{ within } t \text{ steps, where for each } i < t \text{ the active processors are those with index } 0, \ldots, g(i) - 1\}.$$

Let M be an OCRAM, P a functional of rank (1,1), f a function and x, t integers. If $Q \subseteq \mathbf{N}$ then define

$$f_Q(u) = \begin{cases} f(u) & \text{if } u \in Q \\ 0 & \text{else.} \end{cases}$$

Define $\mathcal{M} = \langle M, P \rangle$ to be a *fully specified* OCRAM if for all f, x, t the OCRAM M on input f, x either is halted at step t or executes at step t with active processors $0, \ldots, P(|f_{Q_t}|, |x|) - 1$ where

$$Q_t = Q(M, f, x, t, P(|f_{Q_{t-1}}|, |x|))$$

is the collection of queries made by $\mathcal{M}$ before step t.

If $\mathcal{M} = \langle M, P \rangle$ is a fully specified OCRAM with input f, x define

$$Q_{\mathcal{M}}(f,x,t) = \{y : \mathcal{M} \text{ queries } y \text{ at time } i < t \text{ on input } f, x\}.$$

In place of stating that $\mathcal{M} = \langle M, P \rangle$ is fully specified, usually M is said to run with processor bound P. If $F(\vec{f}, \vec{x})$ abbreviates $F(f_1, ..., f_m, x_1, ..., x_n)$ and P is a second order polynomial, then $P(|\vec{f}|, |\vec{x}|)$ abbreviates $P(|f_1|, ..., |f_m|, |x_1|, ..., |x_n|)$.

The type 2 analogue of concatenation recursion on notation is given by the following.

Definition 4.15 F is defined from G, H, K by concatenation recursion on notation (CRN) if for all $\vec{f}, \vec{x}, y$,

$$\begin{aligned} F(\vec{f},\vec{x},0) &= G(\vec{f},\vec{x}) \\ F(\vec{f},\vec{x},s_0(y)) &= F(\vec{f},\vec{x},y) * \text{BIT}(H(\vec{f},\vec{x},y),0), \text{ provided that } x \neq 0 \\ F(\vec{f},\vec{x},s_1(y)) &= F(\vec{f},\vec{x},y) * \text{BIT}(K(\vec{f},\vec{x},y),0) \end{aligned}$$

where $*$ denotes concatenation.

Definition 4.16 The type 2 functional H is defined by weak bounded recursion on notation WBRN from G, H_0, H_1, K if

$$\begin{aligned} F(\vec{f},\vec{x},0) &= G(\vec{f},\vec{x}) \\ F(\vec{f},\vec{x},s_0(y)) &= H_0(\vec{f},\vec{x},y,F(\vec{f},\vec{x},y)), \text{ if } n \neq 0 \\ F(\vec{f},\vec{x},s_1(y)) &= H_1(\vec{f},\vec{x},y,F(\vec{f},\vec{x},y)) \\ H(\vec{f},\vec{x},y) &= F(\vec{f},\vec{x},|y|) \end{aligned}$$

provided that $F(\vec{f}, \vec{x}, y) \leq K(\vec{f}, \vec{x}, y)$ holds for all $\vec{f}, \vec{x}, y$.

Definition 4.17 The algebra $\mathcal{A}_0$ is the smallest class of functionals (of type 1 and 2) containing 0,s_0,s_1,I,BIT,$|x|$,$\#$, Ap and closed under functional composition, expansion, functional substitution and CRN. The algebra $\mathcal{A}$ is the closure of 0,s_0,s_1,I,BIT,$|x|$,$\#$, Ap under functional composition, expansion, functional substitution, CRN and WBRN.

The following theorem is the type 2 analogue of the fact that AC^0 (or equivalently LH) is characterized by the function algebra A_0.

Theorem 4.18 (Clote, Kapron, Ignjatovic [34]) *A functional $F(\vec{f}, \vec{x})$ belongs to $\mathcal{A}_0$ if and only if it is computable on an* OCRAM *in constant time with at most $P(|\vec{f}|, |\vec{x}|)$ many processors, for some second-order polynomial P.*

To provide some intuition for working with the OCRAM, consider the following program for $Ap(f, x) = f(x)$. Recall that $M_{res} = \text{BIT}(M_{op1}, M_{op2})$ is the easily programmed instruction which, for $i = M_{op2}$, computes the coefficient of 2^i in the binary representation of the integer stored in M_{op1}, *provided* that $i < |M_{op1}|$, and otherwise returns the value 'B'. Define "reverse bit" RBIT, where $\text{RBIT}(x, y) = \text{BIT}(x, |x| \dot{-} (y + 1))$ *provided* that $y < |x|$, and 'B' otherwise.
OCRAM program for $Ap(f, x) = f(x)$.

1 M_0 = 0

2 M_1 = processor number

3 M_2 = $*M_1^g$

4 if (M_2 = B) then M_0^g = M_1

```
5  M3 = M0^g ∸ 1  %  M3 = |x| ∸ 1
6  *M1^g = B  % erase global memory
7 *M1^o = M2  %  in Pi, Mi^o = Xi
8  M4 = f([M0...M3])
9  M5 = BIT(M4, M1)
10  if (M5 = B) then M0^g = M1
11 M5 = M0^g  %  M5 = |f(x)|
12 M6 = M5 ∸ (M1 + 1)
13 M7 = BIT(M4, M6)
14 *M1^o = B
15 *M1^g = B
16 *M1^g = M7
17 if (M1 ≥ M5) then M1^g = B
          %  erase trailing 0's
18 HALT % Now Xi = RBIT(f(x), i)
```

The type 2 analogue of Theorem 3.27 was established by the author (in preparation), and strengthens the principal result of [34].

In his attempted proof of the continuum hypothesis, D. Hilbert [70] studied classes of higher type functionals defined by the operations of composition and primitive recursion. Hilbert's general scheme ([70], p. 186) was of the form

$$\mathcal{F}(G, H, 0) = H$$
$$\mathcal{F}(G, H, n+1) = G(\mathcal{F}(G, H, n), n)$$

where $\mathcal{F}, G, H$ are higher type functionals of appropriate types possibly having other parameters not indicated. Illustrating the power of primitive recursion over higher type objects, Hilbert gave a simple higher type primitive recursive definition of the Ackermann function, known not to be primitive recursive. For example, define

$$F(g, 0) = 1$$
$$F(g, n+1) = g(F(g, n)).$$

Then for $n \geq 3$, the principal functions f_n from Definition 3.29 satisfy $f_{n+1}(x) = F(f_n, x)$.

Definition 4.19 The set T_ρ of all finite types is defined inductively as follows:

- $0 \in T_\rho$
- if $\sigma, \tau \in T_\rho$ then $(\sigma \to \tau) \in T_\rho$.

By induction on τ, every type $\rho = (\sigma \to \tau)$ can be written uniquely in the normal form

$$\rho = \rho_1 \to \rho_2 \to \cdots \to \rho_k \to 0$$

when association is to the right and parentheses are dropped. The *level* of a type is defined as follows:

- level $(0) = 0$
- level $(\rho_1 \to \cdots \to \rho_k \to 0) = 1 + \max_{1 \leq i \leq k}\{level(\rho_i)\}$

If F is of type ρ, where $\rho = \rho_1 \to \cdots \to \rho_k \to 0$, then often $F(X_1, \ldots, X_k)$ is written in place of $F(X_1)(X_2)\cdots(X_k)$.

Higher type functional complexity theory is an emerging field. For reasons of space, only references to a few recent papers will be given. In [84], Ker-I Ko surveyed the theory of sequential complexity theory of real valued functions. In [72], H.J. Hoover investigated parallel computable real valued functions. In [42], S. Cook gave a survey of higher type computational approaches, and proved Theorem 4.9. Cook further proposed that any class C of feasible type 2 functionals must satisfy the following two conditions:

1. BFF $\subseteq C \subseteq$ OPT,
2. C is closed under abstraction and application.

In [125] A. Seth defined a class C_2 of type 2 functionals defined by *counter* Turing machines with polynomial bounds, which satisfies the previous conditions, and proved that no recursively presentable class of functionals exists which contains C_2 and satisfies the previous conditions. In [126] Seth further investigated closure conditions for feasible functionals. In [118], J. Royer studied a polynomial time counterpart to the Kreisel-Lacombe-Shoenfield theorem [86].

Complexity theory for functionals of all finite types was initiated by S. Buss, who in [19] introduced a polynomial time analogue of the *hereditarily recursive operations* HRO to define polynomial time functionals of all finite types decorated with runtime bounds. A. Nerode, J. Remmel and A. Scedrov [102] studied a polynomially graded type system. In [52], J.-Y. Girard, A. Scedrov and P. Scott introduced *bounded linear logic*, and proved a normalization theorem which yielded a characterization of a feasible class of type 2 functionals, whose type 1 section is the class of polytime computable functions. In [45], S. Cook and A. Urquhart introduced an analogue of Gödel's system **T** by admitting a recursor for bounded recursion on notation for type 1 objects. Their system PV^ω provided a natural class of polynomial time higher type functionals (called the *basic feasible functionals of higher type*), whose type-2 section of PV^ω is BFF. In [67], V. Harnik extended Cook-Urquhart's functionals to levels of the polynomial time hierarchy. In [44] S. Cook and B. Kapron characterized the higher type functionals in PV^ω by certain kinds of programming language constructs, *typed while* programs and *bounded loop* programs. This kind of characterization was extended by P. Clote, B. Kapron and A. Ignjatovic in [34] to the higher type functionals in NC$^\omega$, relating *bounded loop* programs with higher type parallel complexity classes. In [127] A. Seth extended his definition of *counter* Turing machine to all finite types, thus characterizing PV^ω by a machine model. If one additionally allows dynamic computation of indices of subprograms within this counter Turing machine model, then Seth has conjectured this class to properly contain PV^ω.

In finite model theory, many complexity classes $\mathcal{C}$ have been characterized via word models over a logic as follows: $L \in \mathcal{C}$ iff there is a closed formula ϕ (in a certain logic over a certain signature) for which

$$L = \{w \in \{0,1\}^* : w \models \phi\}.$$

As mentioned in the introduction, though techniques are similar in spirit to those surveyed in this paper, for reasons of space we do not present such results here. Another direction of finite model theory is the investigation of function algebras, as interpreted over finite structures, rather than over **N**. Here Y. Gurevich [59] showed that LOGSPACE "global" functions can be characterized by primitive recursion over finite structures. In [55] A. Goerdt generalized this to prove that type level $k+1$ recursive definitions over finite structures characterize global functions in the class DTIME$(\exp_k(n^{O(1)})$ where $\exp_0(n) = n$ and $\exp_{k+1}(n) = 2^{\exp_k(n)}$.

In [93] D. Leivant and J.-Y. Marion gave various characterizations of PTIME by typed λ-calculi with pairing over an algebra $\mathbf{W}$ of words over $\{0,1\}$. Recently, Leivant and Marion showed how a natural restriction of functional recurrence with substitution generates exactly PSPACE. In a series of papers (see for instance [91]) D. Leivant investigated various tiering schemes of recursion (extensions of safe recursion) and related complexity classes. Such investigations may have some applicability to programming language design. In [104], building on work of H. Schwichtenberg [124], K.-H. Niggl investigated certain subrecursive hierarchies (analogues of primitive recursive) of partial continuous functionals on Scott domains. As evidenced by the articles in the conference proceedings [92], edited by D. Leivant, higher type functional complexity is an exciting area with many interesting theoretical questions, and the possibility of contributing to new programming language features.

5 Acknowledgements

Many thanks to T. Altenkirch, S. Bellantoni, M. Hofmann, H. Schwichtenberg, T. Strahm, and especially N. Danner and K.-H. Niggl for correcting typos and making various suggestions, though of course, the author is solely responsible for any remaining errors. A special note of thanks is due to Sam Buss, Jan Krajíček, Pavel Pudlák, and Gaisi Takeuti, with whom the author has collaborated either directly or indirectly over a period of years. Thanks to I. Mignani for help in preparing the subject and symbol index. During the preparation of this manuscript, support from the National Science Foundation and Volkswagen Foundation is gratefully acknowledged.

References

[1] W. Ackermann. Zum Hilbertschen Aufbau der reelen Zahlen. *Mathematische Annalen*, 99:118–133, 1928.

[2] M. Ajtai. Σ_1^1-formulae on finite structures. *Annals of Pure and Applied Logic*, 24:1 – 48, 1983.

[3] B. Allen. Arithmetizing uniform NC. *Annals of Pure and Applied Logic*, 53(1):1–50, 1991.

[4] G. Asser. Primitive recursive word-functions of one variable. In Egon Börger, editor, *Computaion Theory and Logic*, pages 14–19. Springer-Verlag, 1987. Lecture Notes in Computer Science 270.

[5] L. Babai and L. Fortnow. Arithmetization: a new method in structural complexity theory. *Computational Complexity*, 1:41–66, 1991.

[6] D. Mix Barrington, N. Immerman, and H. Straubing. On uniformity in NC^1. *Journal of Computer and System Science*, 41(3):274–306, 1990.

[7] D.A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences*, 38:150–164, 1989.

[8] P.W. Beame, S.A. Cook, and H.J. Hoover. Log depth circuits for division and related problems. *SIAM Journal on Computing*, 15:994–1003, 1986.

[9] S. Bellantoni. Predicative recursion and computational complexity. Technical Report 264/92, University of Toronto, Computer Science Department, September 1992. 164 pages.

[10] S. Bellantoni. Predicative recursion and the polytime hierarchy. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 15–29. Birkhäuser, 1995.

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

[12] A. Bel'tyukov. A computer description and a hierarchy of initial Grzegorczyk classes. *Journal of Soviet Mathematics*, 20:2280 – 2289, 1982. Translation from Zap. Nauk. Sem. Lening. Otd. Mat. Inst., V. A. Steklova AN SSSR, Vol. 88, pp. 30 - 46, 1979.

[13] J.H. Bennett. *On Spectra*. PhD thesis, Princeton University, 1962. Department of Mathematics.

[14] S. Bloch. Function-algebraic characterizations of log and polylog parallel time. *Computational Complexity*, 4(2):175–205, 1994.

[15] Stephen Bloch, Jonathan Buss, and Judy Goldsmith. Sharply bounded alternation within $\mathcal{P}$. To appear in Proceedings, DMTCS '96; submitted for journal publication, 1995.

[16] R. Boppana and M. Sipser. The complexity of finite functions. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 759–804. Elsevier, MIT Press, 1990. Elsevier (Amsterdam), MIT Press (Cambridge).

[17] A. Borodin. On relating time and space to size and depth. *SIAM Journal on Computing*, 6:733–744, 1973.

[18] S. Buss. *Bounded Arithmetic*, volume 3 of *Studies in Proof Theory*. Bibliopolis, 1986. 221 pages.

[19] S. Buss. The polynomial hierarchy and intuitionistic bounded arithmetic. In A.L. Selman, editor, *Structure in Complexity Theory*, volume 223, pages 77–103. Springer Verlag, 1986. Springer Lecture Notes in Computer Science.

[20] S. Buss. The boolean formula value problem is in ALOGTIME. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, 1987. pp. 123 - 131.

[21] S.R. Buss. The graph of multiplication is equivalent to counting. *Information Processing Letters*, 41:199–201, 1992.

[22] S.R. Buss. Algorithms for boolean formula evaluation and for tree contraction. In P. Clote and J. Krajíček, editors, *Arithmetic, Proof Theory and Computational Complexity*, pages 96–115. Oxford University Press, 1993.

[23] J.-Y. Cai and M.L. Furst. $PSPACE$ survives three-bit bottlenecks. In *Proceedings of 3th Annual IEEE Conference on Structure in Complexity Theory*, pages 94–102, 1988.

[24] A. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the Association of Computing Machinery*, 28:114 – 133, 1981.

[25] A. Chandra, L. J. Stockmeyer, and U. Vishkin. Constant depth reducibility. *SIAM Journal on Computing*, 13:423–439, 1984.

[26] A. Church. An unsolvable problem in elementary number theory. *American Journal of Mathematics*, 58:345 – 363, 1936.

[27] P. Clote. A first order theory for the parallel complexity class NC. Technical Report BCCS-89-01, Boston College, Computer Science Department, January 1989.

[28] P. Clote. $ALOGTIME$ and a conjecture of S.A. Cook. *Annals of Mathematics and Artificial Intelligence*, 6:57–106, 1992.

[29] P. Clote. A time-space hierarchy between P and PSPACE. *Math. Systems Theory*, 25:77–92, 1992.

[30] P. Clote. Polynomial size frege proofs of certain combinatorial principles. In P. Clote and J. Krajíček, editors, *Arithmetic, Proof Theory and Computational Complexity*, pages 162 – 184. Oxford University Press, 1993.

[31] P. Clote. A note on the relation between polynomial time functionals and Constable's class $\mathcal{K}$. In Kleine-Büning, editor, *Computer Science Logic*. Springer Lecture Notes in Computer Science, 1996.

[32] P. Clote. Nondeterministic stack register machines. *Theoretical Computer Science A*, 178:37–76, 1997.

[33] P. Clote. A safe recursion scheme for exponential time. In L. Beklemishev, editor, *Logical Foundations of Computer Science, LFCS'97*. Springer-Verlag, 1997. July 6–12, 1997 in Yaroslavl, Russia.

[34] P. Clote, B. Kapron, and A. Ignjatovic. Parallel computable higher type functionals. In *Proceedings of IEEE 34th Annual Symposium on Foundations of Computer Science*, Nov 3–5, 1993. Palo Alto CA. pp. 72–83.

[35] P. Clote and G. Takeuti. Exponential time and bounded arithmetic. In A.L. Selman, editor, *Structure in Complexity Theory*, volume 223, pages 125–143. Springer Lecture Notes in Computer Science, 1986.

[36] P. Clote and G. Takeuti. Bounded arithmetics for NC, $ALOGTIME$, L and NL. *Annals of Pure and Applied Logic*, 56:73–117, 1992.

[37] P. Clote and G. Takeuti. First order bounded arithmetic and small boolean circuit complexity classes. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 154–218. Birkhäuser Boston Inc., 1995.

[38] P.G. Clote. A sequential characterization of the parallel complexity class NC. Technical Report BCCS-88-07, Department of Computer Science, Boston College, 1988.

[39] P.G. Clote. Sequential, machine-independent characterizations of the parallel complexity classes $ALOGTIME, AC^k, NC^k$ and NC. In P.J. Scott S.R. Buss, editor, *Feasible Mathematics*, pages 49–70. Birkhäuser, 1990.

[40] A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science II*, pages 24–30. North-Holland, 1965.

[41] R. Constable. Type 2 computational complexity. In *5th Annual ACM Symposium on Theory of Computing*, 1973. pp. 108–121.

[42] S. Cook. Computability and complexity of higher type functions. In Y.N. Moschovakis, editor, *Logic from Computer Science*, pages 51–72. Springer Verlag, 1992.

[43] S. A. Cook. The complexity of theorem proving procedures. In *3rd Annual ACM Symposium on Theory of Computing*, 1971. pp. 151–158.

[44] S.A. Cook and B.M. Kapron. Characterizations of the feasible functionals of finite type. In P.J. Scott S.R. Buss, editor, *Feasible Mathematics*, pages 71–98. Birkhäuser, 1990.

[45] S.A. Cook and A. Urquhart. Functional interpretations of feasibly constructive arithmetic. *Annals of Pure and Applied Logic*, 63(2):pp. 103–200, 1993.

[46] P. Csillag. Eine Bemerkung zur Auflösung der eingeschachtelten Rekursion. *Acta Sci. Math. Szeged.*, 11:169–173, 1947.

[47] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proc. R. Soc. Lond.*, pages 73–90, 1985. Vol. A 400.

[48] R. Fagin. Generalized first–order spectra and polynomial–time recognizable sets. In R. M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings, Vol. 7*, pages 43–73, 1974.

[49] R. Fagin. Finite-model theory—a personal perspective. In S. Abiteboul and P. Kanellakis, editors, *Proc. 1990 International Conference on Database Theory*, pages 3–24. Springer-Verlag Lecture Notes in Computer Science 470, 1990. Journal version to appear in *Theoretical Computer Science*.

[50] S. Fortune and J. Wyllie. Parallelism in random access machines. In *10th Annual ACM Symposium on Theory of Computing*, 1978. pp. 114-118.

[51] M. Furst, J. B. Saxe, and M. Sipser. Parity circuits and the polynomial time hierarchy. *Mathematical Systems Theory*, 17:13–27, 1984. Preliminary version in *Proceedings of the 22nd IEEE Symposium on Foundations of Computer Science*, 1981.

[52] J.-Y. Girard, A. Scedrov, and P. Scott. Bounded linear logic. In P.J. Scott S.R. Buss, editor, *Feasible Mathematics*, pages 195–210. Birkhäuser, 1990.

[53] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. *J. Monat. Math. Phys.*, 38:173 – 198, 1931.

[54] K. Gödel. Conversation with G.E. Sacks. Institute for Advanced Study, 1975.

[55] A. Goerdt. Characterizing complexity classes by general recursive definitions in higher types. *Information and Computation*, 101(2):202–218, 1992.

[56] L. Goldschlager. Synchronous parallel computation. Technical Report 114, University of Toronto, December 1977. 131 pages.

[57] L. Goldschlager. A unified approach to models of synchronous parallel machines. *Journal of the Association of Computing Machinery*, 29(4):pp. 1073–1086, October 1982.

[58] A. Grzegorczyk. Some clases of recursive functions. *Rozprawy Matematyczne*, 4, 1953.

[59] Y. Gurevich. Algebras of feasible functions. In *Proceedings of 24th IEEE Symposium on Foundations of Computer Science*, 1983. pp. 210-214.

[60] Y. Gurevich and S. Shelah. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.

[61] Y. Gurevich and S. Shelah. Nearly linear time. *Symposium on Logical Foundations of Computer Science*, Springer Lecture Notes in Computer Science(363):108–118, 1989. Pereslavl-Zalessky, USSR.

[62] P. Hájek and P. Pudlák. *Metamathematics of first order arithmetic*. Springer-Verlag, 1992.

[63] W. Handley, J. B. Paris, and A. J. Wilkie. Characterizing some low arithmetic classes. In *Theory of Algorithms*, pages 353 – 364. Akademie Kyado, Budapest, 1984. Colloquia Societatis Janos Bolyai.

[64] W.G. Handley. LTH plus nondeterministic summation mod M_3 yields ALINTIME. Submitted, 22 December 1994.

[65] W.G. Handley. Deterministic summation modulo $\mathcal{B}_n$, the semi-group of binary relations on $\{0, 1, \ldots, n-1\}$. Submitted, May 1994.

[66] G.H. Hardy. A theorem concerning the infinite cardinal numbers. *Q.J. Math.*, pages 87–94, 1904.

[67] V. Harnik. Provably total functions of intuitionistic bounded arithmetic. *Journal of Symbolic Logic*, 57(2):466–477, 1992.

[68] K. Harrow. Small Grzegorczyk classes and limited minimum. *Zeit. Math. Logik*, 21:417–426, 1975.

[69] K. Harrow. Equivalence of some hierarchies of primitive recursive functions. *Zeit. Math. Logik*, 25:411–418, 1979.

[70] D. Hilbert. Über das Unendliche. *Mathematische Annalen*, 95:161–190, 1925.

[71] W.D. Hillis and G.L. Steele, Jr. Data parallel algorithms. *Communications of the ACM*, pages 1170–1183, December 1986. 29 (12).

[72] H.J. Hoover. Computational models for feasible real analysis. In S.R. Buss and P.J. Scott, editors, *Feasible Mathematics*, pages 221–238. Birkhäuser, 1990.

[73] N. Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16:760–778, 1987.

[74] N. Immerman. Expressibility and parallel complexity. *SIAM J. Comput.*, 18(3):625–638, 1989.

[75] J. Johannsen. *Schwache Fragmente der Arithmetik und Schwellwertschaltkreise beschränkter Tiefe*. PhD thesis, Universität Erlangen-Nürnberg, 13 May 1996.

[76] J.P. Jones and Y. Matijasivič. A new representation for the symmetric binomial coefficient and its applications. *Ann. Sc. Math., Quebec*, 6(1):81–97, 1982.

[77] N.D. Jones and A.L. Selman. Turing machines and the spectra of first-order formulas. *Journal of Symbolic Logic*, 39:139–150, 1974.

[78] L. Kálmar. Egyszerü példa eldönthetetlen aritmetikai problémára. *Mate és Fizikai Lapok*, 50:1–23, 1943. [In Hungarian with German abstract].

[79] R. Kannan. Towards separating nondeterministic time from deterministic time. In *Proceedings of 22nd IEEE Symposium on Foundations of Computer Science*, 1981. 235-243.

[80] B. Kapron and S. Cook. A new characterization of type-2 feasibility. *SIAM J. Comput.*, 25(1):117–132, 1996. Preliminary version appeared in *IEEE Symposium on Foundations of Computer Science*, 342–347 (1991).

[81] R.M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 871–942. Elsevier, MIT Press, 1990. Elsevier (Amsterdam), MIT Press (Cambridge).

[82] S.C. Kleene. General recursive functions of natural numbers. *Math. Ann.*, 112:727–742, 1936.

[83] S.C. Kleene. Lambda-definability and recursiveness. *Duke Mathematical Journal*, 2:340–353, 1936.

[84] Ker-I Ko. Applying techniques of discrete complexity theory to numerical computation. In R.V. Book, editor, *Studies in Complexity Theory*, pages 1–62. John Wiley and Sons, Inc, 1986.

[85] J. Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory*. Cambridge University Press, 1995.

[86] G. Kreisel, D. Lacombe, and J.R. Shoenfield. Partial recursive functionals and effective operations. In A. Heyting, editor, *Constructivity in Mathematics: Proceedings of a colloquium held in Amsterdam*, pages 195–207. North Holland, 1957.

[87] M. Kutyłowski. Finite automata, real time processes and counting problems in bounded arithmetics. *Journal of Symbolic Logic*, 53(1):243–258, 1988.

[88] D. Leivant. Stratified polymorphism. In *Proceedings of IEEE 4th Annual Symposium on Logic in Computer Science*, pages 39–47, 1989. Journal version: Finitely stratified polymorphism, Information and Computation 93 (1991) 93–113.

[89] D. Leivant. A foundational delineation of computational feasibility. In *Proceedings of IEEE 6th Annual Symposium on Logic in Computer Science*, 1991.

[90] D. Leivant. Stratified functional programs and computational complexity. In *Conference Record of the Twentieth Annual ACM Symposium on Principles of Programming Languages*, 1993.

[91] D. Leivant. Ramified recurrence and computational complexity I: word recurrence and poly-time. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 320–343. Birkhäuser, 1994.

[92] D. Leivant. *Logic and Computational Complexity.* Springer Verlag, 1995. Lecture Notes in Computer Science 960.

[93] D. Leivant and J.-Y. Marion. Lambda-calculus characterizations of poly-time. *Fundamenta Informaticae*, 19:167–184, 1993.

[94] J.C. Lind. Computing in logarithmic space. Technical Report Project MAC Technical Memorandum 52, Massachusetts Institute of Technology, September 1974.

[95] S.S. Marčenkov. A superposition basis in the class of Kalmar elementary functions. *Matematicheskie Zametki*, 27(3):321–332, 1980. Translation in *Mathematical Notes of the Academy of Sciences of the USSR*, Plenum Publishing Company.

[96] K. Mehlhorn. Polynomial and abstract subrecursive classes. *Journal of Computer and System Science*, 12:147–178, 1976.

[97] A.R. Meyer and D. Ritchie. The complexity of loop programs. *Proc. ACM Nat. Conf.*, pages 465–469, 1967.

[98] B. Monien. A recursive and grammatical characterization of exponential time languages. *Theoretical Computer Science*, 3:61–74, 1977.

[99] S.S. Muchnick. The vectorized Grzegorczyk hierarchy. *Zeit. Math. Logik.*, 22:441–80, 1976.

[100] Helmut Müller. *Klassifizierungen der primitiv rekursiven Funktionen.* PhD thesis, Universität Münster, 1974.

[101] V.A. Nepomnjascii. Rudimentary predicates and turing calculations. *Dokl. Akad. Nauk SSSR*, 195:29–35, 1970. Translated in *Soviet Math. Dokl.* **11** (1970) 1462-1465.

[102] A. Nerode, J. Remmel, and A. Scedrov. Polynomially graded logic I – a graded version of system T. In *Proceedings of IEEE 4th Annual Symposium on Logic in Computer Science*, 1989.

[103] A.P. Nguyen. A formal system for linear space reasoning. Technical Report 300/96, University of Toronto, 1996.

[104] K.-H. Niggl. Subrecursive hierarchies on Scott domains. *Archive for Mathematical Logic*, 32:239–257, 1993.

[105] M.J. O'Donnell. *Equational logic as a programming language.* M.I.T. Press, 1985.

[106] J. Otto. Tiers, tensors, and Δ_0^0. Talk at meeting LCC, Indianapolis, organizer D. Leivant, October 13–16 1994.

[107] J. Otto. Half tiers and linear space (and time). Talk at DIMACS Workshop on Computational Complexity and Programming Languages, organized by B.M. Kapron and J. Royer, July 25–26 1996.

[108] J. Otto. *Complexity Doctrines.* PhD thesis, Department of Mathematics and Statistics, McGill University, June 13, 1995.

[109] S.V. Pakhomov. Machine independent description of some machine complexity classes (in Russian). *Issledovanija po konstrukt. matemat. i mat. logike*, VIII:176–185, LOMI 1979.

[110] J. B. Paris and A. J. Wilkie. Counting problems in bounded arithmetic. In C. A. di Prisco, editor, *Methods in Mathematical Logic*, pages 317 – 340. Springer Verlag Lecture Notes in Mathematics, 1983. Proceedings of Logic Conference held in Caracas, 1983.

[111] R. Péter. Über die mehrfache Rekursion. *Mathematische Annalen*, 113:489–526, 1936.

[112] F. Pitt. $\widehat{N}_0$ and $ALOGTIME$. typeset manuscript, 1995.

[113] P. Pudlák. Complexity theory and genetics. In *Proceedings of 9th Annual IEEE Conference on Structure in Complexity Theory*, 1994.

[114] R.W. Ritchie. Classes of predictably computable functions. *Trans. Am. Math. Soc.*, 106:139–173, 1963.

[115] J. Robinson. Definability and decision problems in arithmetic. *Journal of Symbolic Logic*, 14:98–114, 1949.

[116] R.M. Robinson. Primitive recursive functions. *Bulletin of the Amer. Math. Society*, 53:923–943, 1947.

[117] H. E. Rose. *Subrecursion: Function and Hierarchies*, volume 9 of *Oxford Logic Guides*. Clarendon Press, Oxford, 1984. 191 pages.

[118] J.S. Royer. Semantics vs. syntax vs. computation. Typescript, November 29, 1994.

[119] W.L. Ruzzo. On uniform circuit complexity. *J. Comput. System Sci.*, 22:pp. 365–383, 1981.

[120] W. J. Savitch. Relationship between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177 – 192, 1970.

[121] C. P. Schnorr. Satisfiability is quasilinear complete in NQL. *Journal of the Association of Computing Machinery*, 25(1):136–145, 1978.

[122] H. Scholz. Ein ungelöstes Problem in der symbolischen Logik. *Journal of Symbolic Logic*, 17:160, 1952.

[123] H. Schwichtenberg. Rekursionszahlen und die Grzegorczyk-Hierarchie. *Arch. Math. Logik.*, 12:85–97, 1969.

[124] H. Schwichtenberg. Primitive recursion on the partial continuous functionals. In M. Broy, editor, *Informatik und Mathematik*, pages 251–259. Springer-Verlag, 1991.

[125] A. Seth. There is no recursive axiomatization for feasible functionals of type 2. In *Proceedings of IEEE 7th Annual Symposium on Logic in Computer Science*, 1992. pp. 286–295.

[126] A. Seth. Some desirable conditions for feasible functionals of type 2. In *Proceedings of IEEE 8th Annual Symposium on Logic in Computer Science*, 1993.

[127] A. Seth. Turing machine characterizations of feasible functionals of all finite types. In P. Clote and J. Remmel, editors, *Feasible Mathematics II*, pages 407–428. Birkhäuser, 1994.

[128] Y. Shiloach and U. Vishkin. Finding the maximum, merging and sorting in a parallel computation model. *Journal of Algorithms*, 3:57–67, 1982.

[129] P. Shor. Algorithms for quantum computation: discrete log and factoring. In *Proceedings of IEEE 35th Annual Symposium on Foundations of Computer Science*, 1994.

[130] Harold Simmons. The realm of primitive recursion. *Archive for Mathematical Logic*, 27:177–188, 1988.

[131] Th. Skolem. Begründung der elementaren Arithmetik durch die rekurrierende Denkweise ohne Anwendung scheinbarer Veränderlichen mit unendlichem Ausdehnungsbereich. *Skrifter utgit av Videnskapsselskapet, I. Mate. Klasse*, 6, 1923. Oslo.

[132] R. Smullyan. *Theory of Formal Systems*. Annals of Mathematical Studies, no. 47. Princeton University Press, 1961.

[133] L. Stockmeyer and U. Vishkin. Simulation of parallel random access machines by circuits. *SIAM Journal on Computing*, 13:409–422, 1984.

[134] G. Takeuti. Frege proof system and TNC^0. In D. Leivant, editor, *Logic and Computational Complexity*, pages 221–252. Springer Verlag, 1995. Lecture Notes in Computer Science 960.

[135] D.B. Thompson. Subrecursiveness: machine independent notions of computability in restricted time and storage. *Math. Systems Theory*, 6:3–15, 1972.

[136] M. Townsend. Complexity for type-2 relations. *Notre Dame Journal of Formal Logic*, 31:241–262, 1990.

[137] A.M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. Lond. Math. Soc., Series 2*, 42:230–265, 1936-37.

[138] L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[139] P. van Emde Boas. Machine models and simulations. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 1–66. Elsevier, MIT Press, 1990. Elsevier (Amsterdam), MIT Press (Cambridge).

[140] H. Vollmer and K. Wagner. Recursion theoretic characterizations of complexity classes of counting functions. *Theoretical Computer Science* **163** (1996) 245–258.

[141] K. Wagner. Bounded recursion and complexity classes. In *Lecture Notes in Computer Science*, volume 74, pages 492–498. Springer-Verlag, 1979.

[142] K. Wagner and G. Wechsung. *Computational Complexity*. Reidel Publishing Co., 1986.

[143] A.J. Wilkie. Modèles non standard de l'arithmétique, et complexité algorithmique. In A.J. Wilkie and J.-P. Ressayre, editors, *Modèles non standard en Arithmétique et en Théorie des ensembles*, pages 5–45. Publications Mathématiques de l'Université Paris VII, 1983.

[144] A. Woods. Bounded arithmetic formulas and Turing machines of constant alternation. In J.B. Paris, A.J. Wilkie, and G.M. Wilmers, editors, *Logic Coloquium 1984*. North Holland, 1986.

[145] C. Wrathall. Complete sets and the polynomial time hierarchy. *Theoretical Computer Science*, 3:23 – 33, 1976.

Symbol Index

$Q_{\mathcal{M}}(f,x,t)$, 59
$\Box_i^P$, 49
ALOGTIME, 51
DTIMESPACE$(T(n),S(n))$, 6
PTIME, 49
POLYLOGTIME, 51
$c_P(\vec{x})$, 16
$(\text{NORMAL} \cap [\mathcal{F};\mathcal{O}])_*$, 45
$(\mathcal{E}f)_*$, 30
$(\mathcal{R}_2)_*$, 40
$(\mathcal{R}_3)_*$, 40
$(\mathcal{R}_n)_*$, 40
A_0, 18, 24
$A_M(\vec{f},\vec{x},t)$, 53
A_{lmax}, 56
Ap, 54
B, 45
$BFF(f)$, 39
B_f, 11
C_2, 61
$E(z,w;x)$, 46
$FP(f)$, 39
G_f, 11
$H(x)$, 51
I, 17
I_k^n, 17
$I_j^{n,m}$, 45
$K(;a,b,c)$, 50
$K(f)$, 39
$L(M)$, 5
$L_2(QF^+)$, 45
$Lin\mathcal{FC}$, 11
$Log\mathcal{FC}$, 11
NC, 39
$O(f)$, 5
PV^ω, 62
$Pr(;a)$, 50
$Q(M,f,x,t,g)$, 59
$QA_M(\vec{f},\vec{x},t)$, 53
$Q_M(\vec{f},\vec{x},t)$, 53
$Q_M(f,x,t)$, 55
$S(;a)$, 50
$S(t)$, 53
$S_0(;a)$, 45
$S_1(;a)$, 45
T_ρ, 61
$[\mathcal{X};\text{OP}]$, 16
$\#P$, 40
AC^k, 15
ACC(2), 28
ACC(6), 28
ACC(k), 15
BH, 51
BH(x), 41
$\Box_i^P$, 38
Δ_0, 35
Δ_0 formula, 35
FH, 51
FH(x), 41
Γ^N, 37
LSP, 20
MAP, 41
MSP, 20
NC, 15, 50, 51
NC^0, 51
NC^k, 15, 29
NC^ω, 62
$\Omega(f)$, 5
PBRN, 46
SHL, 41
SRN, 45
Σ_0^P, 7
Σ_{n+1}^P, 7
$\Sigma_n - \text{TIME}(T(n))$, 37
TC^0, 15, 26
TL, 21
TR, 21
$\Theta(f)$, 5
ALINTIME, 38, 40
$\alpha \vdash_M \beta$, 5
$*$, 19
β, 22
$\beta(i,z)$, 23
BIT, 18
BS, 22
CA, 35, 37
ETIME, 43
$\mathcal{F}$LH, 23, 24
$\mathcal{F}$LINSPACE, 31
$\mathcal{F}$PTIME, 45
$\mathcal{F}$PSPACE, 32
$\mathcal{G}$CA, 35
$\leq_T$, 52
$\leq_T^P$, 52
$\leq_T^{NP}$, 7
$\leq_T^P$, 7
$\log^{(n)} x$, 11
LTH, 37

$\text{LTH} \subseteq \mathcal{E}^0_*$, 30
maxquery^c_M, 56
$\text{DTimeSpace}(T, S)$, 34
$\text{DTimeSpace}(n^{O(1)}, O(n))$, 34
$\text{SizeDepth}(S(n), D(n))$, 14
$\text{TimeProc}(T(n), P(n))$, 13
$\text{ATIME}(n^{O(1)})$, 33
$\text{BFF}(X)$, 54
$\text{DSPACE}(n \cdot (\log^{(k)} n)^{O(1)})$, 33
INITIAL_M, 21
$\text{MOD}_{k,n}$, 14
NEXT_M, 21
$\text{NORMAL} \cap [\mathcal{F}; \mathcal{O}]$, 45
NP, 6
$\text{NSPACE}(S(n))$, 6
$\text{NTIME}(T(n))$, 6
PSPACE, 33
RF_*, 35
$\text{TH}_{k,n}$, 14
MOD2, 18
$\dot{-}$, 12
μB, 49
μB_i, 49
μFP_i, 38
$\text{NSPACE}(O(\log(n)))$, 37
$\text{NTimeSpace}(T(n), S(n))$, 6
$\oplus$, 14
π_1, 21
π_2, 21
$\rho = (\sigma \to \tau)$, 61
$\tau(x, y)$, 21
τ_n, 21
$\vdash_M$, 6
$\vdash^*_M$, 5
$\vdash^n_M$, 5
cond, 19
*cond*0, 18
*cond*1, 18
*cond*2, 18
$d(P)$, 55
$f^{(n)}(x)$, 11
f_0, 29
f_1, 29
f_2, 29
f_3, 29
f_Q, 55
f_{n+1}, 29
$initial_M$, 21
k-BR, 33
$k -$ BRN, 28
lsp, 20
m-counting, 40
msp, 20
$next_M$, 21
ones, 18
pad, 18
$q_?$, 7
q_A, 4
q_R, 4
q_{no}, 7
q_{yes}, 7
*rev*0, 18
rk_O, 16
$rk_{O,R}$, 16
$s(x)$, 17
$s_0(x)$, 17
$s_1(x)$, 17
$sg, \overline{sg}$, 18
xBy, 20
xEy, 20
xPy, 20
$x\#y$, 18
$\mathcal{A}$, 60
$\mathcal{A}_0$, 60
$\mathcal{CW}_k(\mathcal{F})$, 40
$\mathcal{CW}_k(\mathcal{F})_*$, 40
$\mathcal{E}^2$, 50
$\mathcal{E}^2_* = \text{LINSPACE}$, 30
$\mathcal{E}^2_*$, 32
$\mathcal{E}^3_*$, 39
$\mathcal{E}^k$, 32
$\mathcal{E}^n$, 50
$\mathcal{E}^n_*$, 30
$\mathcal{E}^n$, 29
$\mathcal{E}f$, 29
$\mathcal{FC}$, 11
$\mathcal{F}_*$, 16
$\mathcal{GC}$, 11
$\mathcal{H}_n$, 30
$\mathcal{K}$, 39
$\mathcal{K}(f)$, 39
$\mathcal{M}^1_*$, 35
$\mathcal{M}^2$, 35, 37
$\mathcal{M}^2_*$, 35, 37
$\mathcal{M}^n$, 34
$\mathcal{PR}$, 17, 42
$\mathcal{PR}_1$, 17
$\mathcal{R}_k$, 40
$\mathcal{W}_2(\mathcal{F})_*$, 40
$\mathcal{W}_3(\mathcal{F})_*$, 40
$\mathcal{W}_k(\mathcal{F})$, 40
$\mathcal{W}_k(\mathcal{F})_*$, 40
ℓh, 23
ALINTIME, 8
ALOGTIME, 8
APOLYLOGTIME, 8
$\text{ASPACE}(S(n))$, 8

ATIME($T(n)$), 8
DSPACE($S(n)$), 6
DTIME($T(n)$), 6
ETIME, 6
EXPTIME, 6
PSPACE, 6
PTIME, 6
P, 6
ATM, 7
CA, 35
COMP, 17
CRN, 17
$\mathcal{F}$ALOGTIME, 28
$\mathcal{F}$LOGSPACE, 37
$\mathcal{F}$LTH, 37
$\mathcal{F}$PTIME, 26, 29, 48
$\mathcal{F}$PH, 38
$\mathcal{F}$PSPACE, 29, 33
LINSPACE, 32
LTH, 37
ADD, 17
BRN, 26
NTM, 6
SWRN, 51
RATM, 6
W, 62
GPTIME, 27
k-BTRN, 41
ATM, 7
BFF, 54, 55
BMIN, 30
BPROD, 38
BR, 29
BSUM, 38
BVR, 42
CA, 35
CRAM, 11
CRCW, 11
CREW, 11
CRN, 59
DCL, 15
EREW, 11
ETIME, 50
ITER, 17
LH, 8
LOGSPACE, 37
LRN, 54
LTH, 8
MAJ, 26
NC, 41
NLT, 34
OCRAM, 58, 60
OPT, 53
OTM, 52
PBBR, 41
PH, 8
PID, 11
POTM, 53
PRAM, 11
PRES, 35
PR, 17
PSPACE, 62
QL, 34
RF, 34
RTM, 34
SBBITSUM, 26
SBMAX, 19, 20
SBMIN, 19, 20
SBPROD, 38
SBRN, 27
SBSUM, 38
SCOMP, 45
SDCR, 51
SMIN, 49
SR, 50
TM, 4
VR, 42
VSDCR, 51
WBPR, 33
WBRN, 28, 60
`LSP`, 12
`MSP`, 12

C(;A,B,C), 45
COND, 18

P(;A), 45

Subject Index

k-bounded recursion, 33
k-bounded recursion on notation, 28
k-bounded tree recursion on notation, 41
k-function, 40
rev, 18
(constructive arithmetic, 35
$\mathcal{F}$LOGSPACE, 27
POLY, 54
LOGTIME-uniformity, 15

accepted, 6
accepting computation tree, 8
accepts, 5, 8, 14
alternating multitape Turing machine, 7
alternating Turing machine, 7
answer set, 53
arbitrary fan-in, 14
arithmetizations, 16

back half, 41, 51
basic feasible functionals, 54
basic feasible functionals of higher type, 62
binary successor functions, 17
bitgraph, 11
block size, 22
Boolean circuits, 14
boolean formula valuation problem, 10
bounded 2-value recursion, 42
bounded fan-in, 14
bounded linear logic, 62
bounded loop programs, 62
Bounded minimization, 34
bounded minimization, 30, 34
bounded product, 38
bounded quantifier formulas, 35
Bounded recursion, 29
Bounded recursion on notation, 26
bounded recursion on notation, 2, 26
bounded shift left function, 41
bounded summation, 38
Branching instructions, 30

characteristic function, 16
circuit, 14
Circuit families, 14
commutator, 9
commutator subgroup, 9
complete, 9
composition, 17
computational tree, 6
concatenation function, 19
concatenation recursion on notation, 17, 59
concurrent random access machine, 11
conditional function, 18
configuration, 4, 21
constructive arithmetic, 2
control variables, 45
counter Turing machine, 62
counter Turing machines, 61
course-of-values recursion, 38, 42

decided, 5
definition by cases, 19
depth, 14, 55
descriptive complexity theory, 3
direct connection language, 15
Divide and conquer, 38
divide and conquer recursion, 29

elementary functions, 2
expansion, 53
extended rudimentary, 2

fan-in, 14
fan-out, 14
feasible type 2 functionals, 52
finite basis, 39
finite types, 61
front half, 41, 51
fully specified OCRAM, 59
function algebra, 16
function length cost, 53, 55
functional complexity theory, 52
functional composition, 53
functional substitution, 54

gates, 14
genetic Turing machine, 11
global read/write, 11
global shared memory, 11
graph, 11
Grzegorczyk's classes, 29

half function, 51
halted configuration, 4
Heinemann hierarchy, 30

hereditarily recursive operations, 62
higher type parallel complexity classes, 62

in-degree, 14
Incremental instructions, 30
index answer tape, 6
index query tape, 6
iteration, 17

least significant part function, 20
length, 55
limited recursion on notation, 54
linear growth, 11
linear time hierarchy, 8, 35
logarithmic growth, 11
logtime hierarchy, 8

makeindex, 1
modular counting gate, 14
most significant part function, 20
multiple bounded recursion on notation, 44

nearly linear time, 34
next configuration, 5
nondeterministic multitape Turing machine, 6
normal, 44

operation, 16
operator, 16
oracle answer tape, 53
oracle concurrent random access machine, 58
oracle query state, 53
oracle query tape, 53
oracle registers, 58
oracle Turing machine, 52
oracle query tape, 7
oracle Turing machine, 7
out-degree, 14
output node, 14

pairing function, 21
Parallel machine model, 11
parallel random access machine, 11
parity gate, 14
polynomial growth, 11
polynomial time hierarchy, 8
polynomial time oracle Turing machine, 53
polynomially bounded branching recursion, 41
polynomially bounded recursion on notation, 46
positive extended rudimentary, 2
Presburger arithmetic, 35
Presburger definable sets, 35
primitive recursion, 17
principal, 29
priority resolution, 12
probabilistic Turing machine, 11
projection functions, 17

quantum Turing machine, 11
quasilinear space, 34
quasilinear time, 34
query set, 53

ramified recurrence, 50
random access, 6
rank, 16
rank (k, ℓ), 52
regular, 8
rejected, 8
repeated squaring, 35
Ritchie-Cobham property, 54
rudimentary, 2
rudimentary functions, 34
rudimentary sets, 35

safe, 44, 45
safe composition, 45
safe divide and conquer recursion, 51
safe minimization, 49
Safe recursion, 44
safe recursion, 50, 62
safe recursion on notation, 45
safe weak recursion on notation, 51
safe-storage Turing machines, 33
Scott domains, 62
second order polynomials, 55
second-order polynomial, 60
Sharply bounded maximization, 19
sharply bounded minimization, 19
sharply bounded product, 38
sharply bounded quantifier, 19
sharply bounded recursion on notation, 27
sharply bounded summation, 38
simultaneous bounded recursion, 42
simultaneous bounded recursion on notation, 42
simultaneous recursion, 42
simultaneous recursion on notation, 42
size, 14
solvable, 9

space $S(n)$, 5, 6, 8
stack register machine, 30
stack registers, 30
Storage instructions, 30
successor function, 17
syntactic monoid, 9

threshold gate, 14
tiering, 45, 62
time $T(n)$, 5, 8
time $T(n)$, 6
transition function, 4, 7
transition relation, 6
Turing machine, 4
Turing machines, 4
type 2 functional, 52
Type 2 functionals, 52
type level, 61
typed while programs, 62

unbounded, 14
unit cost, 53
unit cost model, 54

vectorized Grzegorczyk classes, 30
very safe divide and conquer recursion, 51

weak bounded primitive recursion, 33
weak bounded recursion on notation, 28, 60
well quasi-ordering, 54
work register, 30

yield, 5