Title	計算の複雑さに関する構造的研究
Author(s)	藤澤，孝敏
Citation	
Issue Date	2000-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/1324
Rights	
Description	Supervisor:石原 哉，情報科学研究科，修士

Structural Computational Complexity Theory

By Takatoshi Fujisawa

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Associate Professor Hajime Ishihara

February 15, 2000

Contents

Chapter 1

Introduction

Theoretical computer science is the mathematical study of models of computation. As such, it originated in the 1930's, well before the existence of modern computers, in work of the mathematical logicians. They would like to grips with the notion *effective computation.* They knew various algorithms for computing things effectively, but they weren't quite sure how to define *effective computable* in a general way that would allow them to distinguish between the computable and the noncomputable. Several alternative formalisms evolved, each with its own peculiarities, in the attempt to nail down this notion:

- Turing machines (Alan Turing);
- Post systems (Emil Post);
- μ-recursive functions (Stephen C. Kleene, Kurt Gödel, Jacques Herbrand);
- λ-calculus (Alonzo Church); and
- combinatory logic (Moses Schönfinkel, Haskell B. Curry).

All of these systems embody the idea of *effective computation* in one form or another. They work on various types of data; for example, Turing machines treat strings over a finite alphabet, μ-recursive functions manipulate the natural numbers, the λ-calculus handles λ-terms, and combinatory logic manipulates terms built from combinator symbols.

Because these vastly dissimilar formalisms are all computationally equivalent, the common notion of computability that embody is extremely robust, which is to say that it is invariant under fairly radical perturbations in the model. All these mathematical logicians with their pet systems turned out to be looking at the same thing from different angles. This was too striking to be mere coincidence. They soon came to the realization that the commonality among all these systems must be the elusive notion of effective computability that they had sought for so long. Computability is not just Turing machines, nor the λ-calculus, nor the μ-recursive functions, nor the "C" programming language, but the common spirit embodied by them all.

Alonzo Church gave voice to this thought, and it has since become known as the *Church's thesis* (or the *Church-Turing thesis*). It is not a theorem, but rather a declaration that all these formalisms capture precisely our intuition about what it means to be effectively computable in principle, no more and no less. Church's thesis may not seem like such a big deal in retrospect, since by now we are thoroughly familiar with the capabilities of modern computers; but keep in mind that at the time it was first formulated, computers and programming languages had yet to be invented. Coming to this realization was an enormous intellectual leap.

One would discover that the resources required for computing computable function is finite. No one knows how much time or memory is required. However if a function is computable in principle, a function is not always computable feasibly. Computer problems come in different varieties; some are easy and some hard. For example, the sorting problem is an easy one. Say that you need to arrange a list of numbers in ascending order. Even a small computer can sort a million numbers rather quickly. Compare that to a scheduling problem. Say that you must find a schedule of class for the entire university to satisfy some reasonable constraints, such as that no two classes take place in the same room at the same time. The scheduling problem seems to be much harder than the sorting problem. If you have just a thousand classes, finding the best schedule may require centuries, even with a supercomputer. *Computational complexity* deal with the cost or difficulty of computing the computable functions. The field of *structural computational complexity* defines the class corresponding to the difficulty, and study the including relation of some classes. By showing difficulty, structural complexity theory is applicable to *cryptography.* Structural complexity theory provides the way to gain evidence for a code's security.

To study structural computational complexity theory, we utilize the methods of providing computable functions. A Turing machine gives the notion *time* and *space* to define complexity class. For instance the class of *polynomial time* computable functions, *logarithm space* computable functions, and so on. Also, to do so, we introduce a few kinds of Turing machines for example the *nondeterministic, oracle, alternating, probabilistic, genetic* Turing machine. Using some recursions, the derivation of primitive recursion is also useful method. By restricting the scheme of primitive recursion to allow only limited summations and limited products, the *elementary functions* were introduced in 1934 by L. Kalmár. In 1953, A. Grzegorczyk studied the class $\mathcal{E}^k$ obtained by closing certain fast growing "diagonal" functions under composition and it bounded recursion or *bounded minimization.* Our 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. A number of machine depending complexity classes have been characterized by function algebra, such as linear space by R.W. Ritchie in 1963, polynomial time by A. Cobham in 1965, logspace by J.C. Lind in 1974, polynomial space by D.B. Thompson in 1972, some significant *circuit* complexity classes by P. Clote in 1990. 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.

Although the author does not treat in this paper, certain important topics still remain. *Quantum computation* would cause new current to the theory of computation. And the another is *type 2 functional* complexity, since many programming languages allow functions to be passed as parameters to other functions or procedures.

The aim of this paper is the establishment of the notion of "feasible" computable. In this paper, we use mainly the method of *recursion schemes* and function algebra is a smallest class of functions containing certain initial functions and closed under certain operations. The class of polynomial time computable functions and Constable's class $\mathcal{K}$ are considered as the feasibly computable class. Then for a quick look at the table of contents, Chapter 2 introduces certain machine models. Turing machines and circuit families are especially mentioned. A Turing machine is a much more accurate model of general computer, and a Turing machine can do everything that a real computer can do. Hence we can understand intuitively the classes by Turing machines. Circuit families are the most simple parallel machine model and the model of electronic devices wired together in a design called a *digital circuit*. In chapter 3, we focus polynomial time computable functions. Historically, Cobham's machine independent characterization of the polynomial time computable functions by *bounded recursion on notation* was the start of modern complexity, theory, indicating a robust and mathematically interesting field. Recently various techniques of characterizing polynomial time computable functions are invented for example *safe recursion on notation* by S. Bellantni and S. Cook in 1992, *ramified recurrence* by D. Leivant in 1993, *full concatenation recursion on notation* by H. Ishihara in 1998, etc. the author explain full concatenation recursion on notation and mention safe recursion on notation. In chapter 4, we discuss the Constable's class $\mathcal{K}$ which defined polynomial analogously Kalmár elementary functions. Despite that $\mathcal{K}$ is very natural for arithmetic, the equivalent class with $\mathcal{K}$ in machine model is not discovered. Then the author would like to find the class corresponding to $\mathcal{K}$.

Note that, in this paper, we follow the notations and the definitions in Clote [6].

Chapter 2

Machine Models

Although there is the enormous diversity of abstract machine models and complexity classes, the author will treat only the most natural and robust models and classes as preliminaries.

2.1 Turing machines

We introduce Turing machines named after Alan Turing who invented them in 1936. Turing machines are invented in the 1930s, long before real computers appeared. However modern computers are based on the notion of the Turing machine. Considering the "computer" as an idealized human clerk, Turing argued that the "behavior of 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 "state of mind" should be finite, since "human memory is necessarily limited". Turing also introduced nondeterministic and oracle Turing machine. 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 is a finite read-only input tape alphabet not containing the blank symbol B,
- Γ is a finite read-write work tape alphabet,
- δ is the transition function and maps

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

  into

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

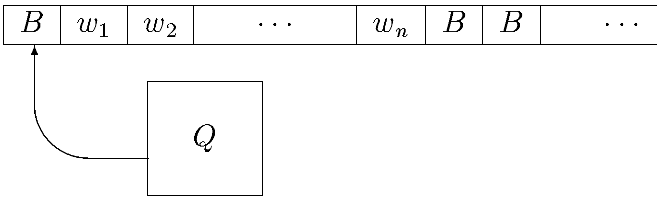


Figure 2.1: Turing machine

A Turing machine is assumed to have a one-way infinite input tapes and k one-way infinite work tape. 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 above.

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 left most 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 Turing machine does the following actions according to the current state and the symbols scanned by tape heads at the same time:

- enters new state,
- rewrites a new symbol on that tape cell, and
- leaves its head at the same position or moves it either left or right one cell.

A Turing machine is different from tow-way finite automata since a Turing machine can rewrite tape symbols.

A *configuration* is a member of $Q \times (\Sigma \cup \{B\})^* \times (\Sigma \cup \{B\})^{*k} \times \mathbf{N}^{k+1}$, and indicates the current state, tape contents, and head positions.[1] Alternately, a configuration can be abbreviated by underscoring the symbols currently 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

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

[1] $A^* = \{x_1 x_2 \cdots x_n \mid n \geq 0 \text{ and } x_i \in A, 1 \leq i \leq n\}$. Note that n can be 0; thus the null string λ is in A^* for any A.

be configurations for M on input x. The 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:
    - $\sigma_i, \tau_i \in \Gamma \cup \{B\}$ and $u_i, v_i, w_i \in (\Gamma \cup \{B\})^*$
    - $\alpha_i = u_i \sigma_i v_i$ and $\beta_i = u_i \tau_i w_i$
    - $|u_i| = n_i$ (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 $1 \leq i \leq k$:
    - $m_i < |\beta_i|$
    - 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-th 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 $(q_0, \underline{B}wB, \underline{B}, \ldots, \underline{B}) \vdash_M^* (q_A, \underline{B}wB, \underline{B}, \ldots, \underline{B})$. 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).

Definition 2.2

$$O(f) = \{g : (\exists c > 0)(\exists n_0)(\forall n \geq n_0)[g(n) \leq c \cdot f(n)]\}$$

Hence $n^{O(1)}$ denotes the set of all polynomially bounded functions. We define some complexity class by TM.

Definition 2.3 If T, S are one-place functions, then[2]

$$\begin{aligned} \text{DTIME}(T(n)) &= \{L \subseteq \Sigma^* : L \text{ accepted by a TM in time } O(T(n))\} \\ \text{DSPACE}(S(n)) &= \{L \subseteq \Sigma^* : L \text{ accepted by a TM in space } O(S(n))\} \end{aligned}$$

[2] Logarithms are with respect to base 2.

$$\begin{aligned}
\text{PTIME} &= P = \text{DTIME}(n^{O(1)}) \\
\text{LOGSPACE} &= \text{DSPACE}(O(\log n)) \\
\text{ETIME} &= \cup_{c\geq 1}\text{DTIME}(2^{c\cdot n}) = \text{DTIME}(2^{O(n)}) \\
\text{EXPTIME} &= \cup_{c\geq 1}\text{DTIME}(2^{n^c}) = \text{DTIME}(2^{n^{O(1)}}).
\end{aligned}$$

Nondeterminism is an important abstraction in computer science. It refers to situations in which the next state of a computation in not uniquely determined by the current state. Nondeterminism is also important in the design of efficient algorithms. The famous $P \neq NP$ problem—whether all problems solvable in nondeterministic polynomial time can be solved in deterministic polynomial time—is major open problem in computer science and arguably one of the most important open problems in all of mathematics.

Definition 2.4 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

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

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, yields and acceptance analogous to previously defined notions. A nondeterministic computation corresponds to a *computational tree* whose root is the initial configuration, whose leaves are halted computations, and whose initial 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 word 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. $O(S(n))$]; NP = NTIME$(n^{O(1)})$.

Definition 2.5 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-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 of the query index tape. Unlike the oracle Turing machine in Definition 2.9, the query index tape is not automatically erased after making an input bit query.

With previous definitions, to scan k-th cell to move, a tape head moves to k-th cell and we require time k. However RATM can scan k-th cell if M writes the binary representation of k and enters special state q_I, then we require time $|k| = \lceil \log(k+1) \rceil$. From now on, unless otherwise indicated, the indicated machine model is RATM.

An *alternating Turing machine* (ATM) is the generalization of NTM. When used with random access, this model allows sublinear runtimes and can be viewed as a kind of 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 2.4 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 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 if an *accepting* configuration,
- if α if universal, then for all β, $\alpha \vdash_M \beta \Rightarrow \beta \in T$, and
- if α if 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 input 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.

Definition 2.7

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

Definition 2.8 The *logtime hierarchy* LH [resp. the *polynomial time hierarchy* PH] is the collection of languages $L \subseteq \Sigma^*$, for which L accepted by an ATM in time $O(\log n)$ [resp. $n^{O(1)}$] with at most $O(1)$ alternations. 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.

Definition 2.9 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.

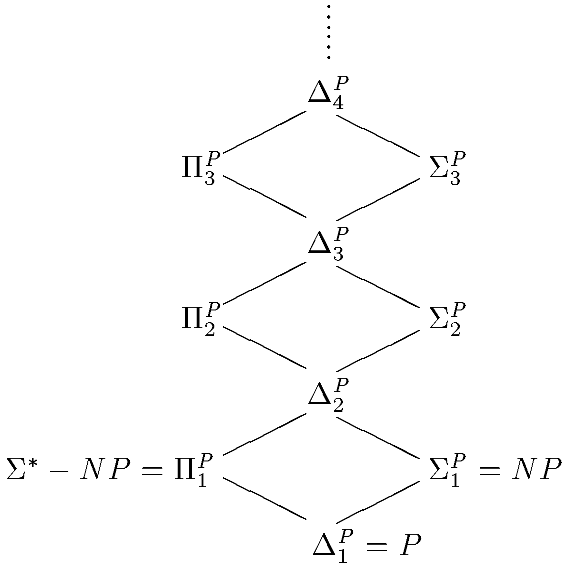


Figure 2.2: Polynomial time hierarchy

Then we show the another equivalent definition of the polynomial time hierarchy(PH). For $A \subseteq \Sigma^*$ and $B \subseteq \Gamma^*$ 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. We define $\Sigma_i^P, \Pi_i^P, \Delta_i^P$ such as,

$$\Sigma_0^P = \Pi_0^P = \Delta_0^P = P$$

$$\Delta^P_{n+1} = \{A : (\exists B \in \Sigma^P_n)(A \leq^P_T B)\}$$
$$\Sigma^P_{n+1} = \{A : (\exists B \in \Sigma^P_n)(A \leq^{NP}_T B)\}$$
$$\Pi^P_{n+1} = \Sigma^* - \Sigma^P_{n+1}.$$

Since $\Delta^P_1 = P$, $\Sigma^P_1 = NP$ then PH is of the figure 2.2.

In the figure 2.2, classes is larger as the upper part. For instance

$$\Pi^P_i \subseteq \Delta^P_{i+1}, \quad \Sigma^P_i \subseteq \Delta^P_{i+1}, \quad \Delta^P_i \subseteq \Sigma^P_{i+1}, \quad \Delta^P_i \subseteq \Pi^P_{i+1}.$$

It is still unknown that whether PH is strict or not and $P \neq NP$ problem is a typical case. However it is known that EXPTIME contains PH properly.

Definition 2.10 A function $f(x,1,\ldots,x_n)$ has *polynomial growth* if

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

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 $\mathcal{FC}$ is the class of functions of polynomial growth whose bitgraph belongs to $\mathcal{C}$.

2.2 Circuit families

Circuit families is the most simple model of the parallel computation machines. In this section, we consider the computation of the circuit families.

Definition 2.11 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.[3] 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.

Definition 2.12 *Boolean circuits* have basis $\wedge, \vee, \neg$, where $\wedge, \vee$ may have fan-in larger than 2 (as described blow, the AC^k [resp. NC^k] model concerns unbounded fan-in [resp. fan-in 2] boolean circuits). A *threshold* gate TH$_{k,n}$ outputs 1 if at least k of its n inputs is 1.

[3] 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-input 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.

Definition 2.13 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

$$f_v(x_1, \ldots, x_n) = g(f_{v_1}(x_1, \ldots, x_n), \ldots, f_{v_m}(x_1, \ldots, x_n)).$$

Definition 2.14 A circuit C_n *accepts* the words $x_1 \cdots x_n \in \{0,1\}^n$ 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.

Definition 2.15 The *depth* of a circuit is the length of the largest path form an input to an output node, while the *size* is the number of gates. A language $L \subseteq \{0,1\}^*$ belongs to $\textsc{SizeDepth}(S(n), D(n))$ 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 $depth(C_n) = O(D(n))$.

Example 2.16 We can easily construct *exclusive-or*. Let

$$x_1 \oplus x_2 = \begin{cases} 0 & \text{if } x_1 = x_2 \\ 1 & \text{if } x_1 \neq x_2 \end{cases}$$

And exclusive-or is explained by $\neg(\neg(x_1 \wedge \neg(x_1 \wedge x_2)) \wedge \neg(\neg(x_1 \wedge x_2) \wedge x_2))$. This binary operator $\oplus$ satisfy the following properties.

- $x \oplus (y \oplus x) = (x \oplus y) \oplus z$ (associativity)
- $x \oplus y = y \oplus x$ (commutativity)

Then *parity* is the n-ary function such that $x_1 \oplus x_2 \oplus \cdots \oplus x_{n-1} \oplus x_n$. If the sum of the inputs $(x_1 + x_2 + \cdots + x_n)$ is even, then $parity(x_1, \cdots, x_n) = 1$, else $parity(x_1, \cdots, x_n) = 0$. A boolean circuit which computes the function $f(x_1, x_2) = x_1 \oplus x_2$ is as in Figure 2.3.

Without a *uniformity* condition, circuit families of depth 2 and size 1 can accept nonrecursive 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 bee suggested, (PTIME-uniformity, LOGSPACE-uniformity, U_{E^*}-uniformity, etc.), but the most robust (and strictest) appears to be that of LOGTIME-uniformity, which is adopted in this paper.

Definition 2.17 (Barrington-Immaerman-Straubing [1]) 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.

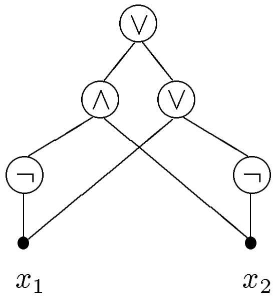


Figure 2.3: Exclusive-or

Definition 2.18 For $k \geq 0$, $\textsc{ac}^k$ [resp. $\textsc{nc}^k$] is the class of languages accepted by $\textsc{logtime}$-uniform $\textsc{SizeDepth}(n^{O(1)}, O(\log^k n))$ over the boolean basis, where $\wedge, \vee$ have arbitrary fan-in [resp. fan-in 2], and $\textsc{nc} = \cup_k \textsc{nc}^k = \cup_k \textsc{ac}^k$ $(= \textsc{ac})$. And $\textsc{tc}^0$ is the class of languages in $\textsc{logtime}$-uniform $\textsc{SizeDepth}(n^{O(1)}, O(1))$ over the boolean basis $\textsc{th}_{k,n}$.

Remark 2.19 In this paper, circuit classes such as $\textsc{ac}^k$, $\textsc{nc}^k$, $\textsc{nc}$, and $\textsc{tc}^0$ sometimes denote both language classes, though more often function classes, where the intended meaning is clear from context. That is, we write $\textsc{nc}$ in place of $\mathcal{F}\textsc{nc}$, 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 [16], using the alternating Turing machine model.

Fact 2.20 *The following containments are known:*

$$\textsc{nc}^k \subseteq \textsc{ac}^k \subseteq \textsc{nc}^{k+1} \textit{ for any } k.$$

Proof. The first inclusion is trivial for the definitions.The second inclusion follows from that we simulate every $\wedge$ and $\vee$ gate of fan-in 2 by increasing the *depth* with $O(\log n)$, and its *size* is bounded by polynomial. ■

None of the inclusions are known to be strict or not without $\textsc{ac}^0 \neq \textsc{nc}^1$.

Theorem 2.21

$$parity(x_1, \cdots, x_n) \in \textsc{nc}^1,$$

$$parity(x_1, \cdots, x_n) \notin \textsc{ac}^0.$$

Chapter 3

Function Algebras

C. Kleene formalized the notion of μ-recursive functions being based on the study by K. Gödel and J. Herbrand. Modern programming languages such as "PASCAL" or "C" can be viewed as an implementation of μ-recursive functions.

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

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

where P is often written in place of c_P. If $\mathcal{F}$ is a class of functions denoted $\mathcal{F} = [f_1, f_2, \ldots; O_1, O_2, \ldots]$, then $\mathcal{F}_*$ is the class of predicates whose characteristic function belongs to $\mathcal{F}$.

Definition 3.2 The *successor* function $s(x) = x + 1$; the n-place projection functions $I^n_k(x_1, \ldots, x_n) = x_k$; I denotes the collection of all projections.

Definition 3.3 The function f is defined by *composition* ($\textsc{comp}$) from the functions $h, g_1, \ldots, g_m$ if

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

Definition 3.4 The function f is defined by *primitive recursion* ($\textsc{pr}$) from functions g, h if

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

The collection $\mathcal{PR}$ of primitive recursive functions is $[0, I, s; \textsc{comp}, \textsc{pr}]$.

Definition 3.5 The function f is defined by *unbounded minimization* (MIN) from the function g, denoted by $f(x_1, \ldots, x_n) = \mu i[g(i, x_1, \ldots, x_n)]$, if

$$f(x_1, \ldots, x_n) = \begin{cases} \min\{i : g(i, x_1, \ldots, x_n) = 0\} & \text{if such exists} \\ \perp & \text{otherwise.} \end{cases}$$

The μ-recursive functions is the algebra $[0, I, s; \text{COMP}, \text{PR}, \text{MIN}]$.

3.1 An algebra for the logtime hierarchy

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, Bennet introduced the scheme of *recursion on notation*, which Cobham [8] later used to characterize the polynomial time computable functions. The scheme of *recursion on notation* requires $|x|$ steps to compute $f(x)$.

Definition 3.6 The *binary successor* functions s_0, s_1 satisfy $s_0(x) = 2 \cdot x$, $s_1(x) = 2 \cdot x + 1$.

Definition 3.7 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

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

This scheme can be written in the abbreviated form

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

The scheme CRN was first introduced by P. Clote [4].

Definition 3.8 The length of x in binary satisfies $|x| = \lceil \log(x+1) \rceil$; $||x||$ is defined as $|(|x|)|$, etc.; $\text{MOD2}(x) = x - 2 \cdot \lfloor \frac{x}{2} \rfloor$; the function $\text{BIT}(i, x) = \text{MOD2}(\lfloor \frac{x}{2^i} \rfloor)$ 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

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

Notice that by finitely many applications of $0, s_0, s_1$ we may suppose that all constant functions belong to A_0. For instance, the constant function 1 can be defined by $s_1(0)$, the constant function 2 can be defined by $s_0(s_1(0))$, and in general the constant function k can be defined by

$$s_{k_0}(s_{k_1}(\ldots s_{k_{n-2}}(s_{k_{n-1}}(0))) \ldots))$$

where the binary representation of k is $k_{n-1}\dots k_0$ ($\mathrm{BIT}(i,k) = k_i$); i.e.

$$k = \sum_{i<n} k_i \cdot 2_i.$$

We define reverse function rev by CRN. The auxiliary reverse function $rev0(x,y)$ gives the $|y|$ many least significant bits of x written in reverse. Let

$$\begin{array}{rcl} rev0(x,0) & = & 0 \\ rev0(x,s_i(y)) & = & s_{\mathrm{BIT}(|y|,x)}(rev0(x,y)). \end{array}$$

The reverse of the binary notation for x is given by $rev(x) = rev0(x,x)$. For instance the integer 14 has binary notation 1110 whose reverse is 111, corresponding to the integer 7, so $rev(14) = 7$.

Let

$$\begin{array}{rcl} ones(0) & = & 0 \\ ones(s_i(x)) & = & s_1(ones(x)) \end{array}$$

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

$$\begin{array}{rcl} pad(x,0) & = & x \\ pad(x,s_i(y)) & = & s_0(pad(x,y)) \end{array}$$

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

$$\begin{array}{rcl} sg(x) & = & \mathrm{BIT}(0, ones(x)) \\ \overline{sg}(x) & = & \mathrm{BIT}(0, pad(1,x)). \end{array}$$

The conditional function

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

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

$$\begin{array}{rcl} cond0(0,y) & = & 0 \\ cond0(s_i(x),y) & = & s_{\mathrm{BIT}(0,y)}(cond0(x,y)) \end{array}$$

$$cond1(x,y) = sg(cond0(x,y))$$

$$\begin{array}{rcl} cond2(x,0) & = & 0 \\ cond2(x,s_i(y)) & = & s_{cond1(x,s_i(y))}(cond2(x,y)) \end{array}$$

so that

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

$$cond1(x,y) = \begin{cases} 0 & \text{if } x = 0 \\ \textsc{bit}(0,y) & \text{else} \end{cases}$$

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

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

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

Then the conditional function *cond* is defined by

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

Example 3.9 With *cond* one can form (characteristic functions of) predicates by applying boolean operations AND, OR, NOT to other predicate.

$$\begin{array}{rcl} \textsc{and} & \cdots & cond(x,0,cond(y,0,1)) \\ \textsc{or} & \cdots & cond(x,cond(y,0,1),1) \\ \textsc{not} & \cdots & cond(x,1,0) \end{array}$$

Additionally, using *cond*, one can introduce functions using *definition by cases*

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

where predicate $P_1, \ldots, P_n$ are disjoint and exhaustive.

The *sharply bounded quantifier* is of the form $(\exists x \leq |y|)$ or $(\forall x \leq |y|)$.

Lemma 3.10 $(A_0)_*$ *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

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

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 3.11 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

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

Lemma 3.12 *$(A_0)_*$ is closed under* SBMIN.

Proof. Let

$$\begin{aligned} k(0,x,\vec{y}) &= 0 \\ k(s_i(z),x,\vec{y}) &= s_{h(z,x,\vec{y})}(k(z,x,\vec{y})) \end{aligned}$$

where

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

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

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

■

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 $B \in A_0$.

We specify useful functions. Define the *most significant part* function MSP by

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

and the *least significant part* function LSP by

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

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

$$msp(x,y) = \lfloor x/2^{|y|} \rfloor = \text{MSP}(x, |y|)$$

$$lsp(x,y) = x \bmod 2^{|y|} = \text{LSP}(x, |y|),$$

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

$$lsp(x,y) = msp(rev(msp(rev(s_1(x)), msp(x,y))), 1).$$

Lemma 3.13 *$(A_0)_*$ is closed under part-of quantifier $(\exists xBy)$.*

Proof. If the predicate $R(x,\vec{z})$ is definable in A_0, then $(\exists xBy)R(x,\vec{z})$ is definable in A_0 as following. Define f by

$$\begin{aligned} f(0,\vec{z}) &= c_{R(0,\vec{z})} \\ f(s_i(x),\vec{z}) &= \begin{cases} s_1(f(x,\vec{z})) & \text{if } R(\textsc{msp}(y,s_i(x)),\vec{z}) \\ s_0(f(x,\vec{z})) & \text{else.} \end{cases} \end{aligned}$$

Now it is clear that $(\exists xBy)R(x,\vec{z})$ iff $sg(f(y,\vec{z}))=1$. ■

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

$$|x| < |y|$$

$$\text{OR}$$

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

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

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 $\textsc{gen}(x,y,z)$ hold iff the $|z|$-th bit of both x and y is 1 and $\textsc{prop}(x,y,z)$ hold iff the $|z|$-th bit of either x of y is 1. Define $carry(x,y,0)=0$ and $carry(x,y,s_i(z))$ to be 1 iff

$$(\exists uBz)[\textsc{gen}(x,y,u) \wedge (\forall uBz)[|v| > |u| \rightarrow \textsc{prop}(x,y,v)]].$$

Then $sum(x,y,z) = x \oplus y \oplus carry(x,y,z)$ where the EXCLUSIVE-OR $x \oplus y$ is defined by $cond(x, cond(y,0,1), cond(y,1,0))$. Modified subtraction

$$x \dot{-} y = \begin{cases} x-y & \text{if } x-y \ge 0 \\ 0 & \text{else} \end{cases}$$

is also belong to A_0 by using the two's complement trick.

Two's complement is as a following.

Example 3.14 We consider the example $9 \dot{-} 4$.

- Flip the bits of the binary representation[1] of 4 with the binary length of 9,

$$4 = 0100_2 \quad \Longrightarrow \quad 1011_2$$

- add 1011_2 and 1, then 1100_2,

[1] If $x = a_2$ then a is the binary representation of x.

- add $9(=1001_2)$ and 1100_2,

$$1001_2 + 1100_2 = 10101_2$$

- and delete left-most bit

$$0101_2 = 5(= 9\dot{-}4).$$

Let

$$\begin{aligned} flip(0) &= 1 \\ flip(s_0(x)) &= s_1(flip(x)) \\ flip(s_1(x)) &= s_0(flip(x)). \end{aligned}$$

Generally

$$x\dot{-}y = \begin{cases} lsp(x + flip(y) + 1, x) & \text{if } x \geq y \\ 0 & \text{else.} \end{cases}$$

For many later applications, pairing and sequence encoding functions are needed. To this end, define the *pairing function* $\tau(x,y)$ by

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

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. For instance, if $x = 4$ and $y = 9$, then $\max(|4|,|9|) = 4$ and so the binary representation of $\tau(4,9)$ is 1010011001. 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

$$\begin{aligned} \pi_1 &= \text{TL}\left(\text{MSP}\left(z, \left\lfloor \frac{|z|}{2} \right\rfloor\right)\right) \\ \pi_2 &= \text{TL}\left(\text{LSP}\left(z, \left\lfloor \frac{|z|}{2} \right\rfloor\right)\right) \end{aligned}$$

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

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

We give following lemma to encoding a sequence efficiently.

Lemma 3.15 (P. Clote [6]) *If $f \in A_0$ then there exists sequence number $g \in A_0$ such that for all x,*

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

Proof. The sequence $(a_1, \ldots, a_{|x|})$ is encoded by $z = \langle a_1, \ldots, a_{|x|}\rangle$. Let

$$k(u) = \begin{cases} 1 & \text{if } |u| \bmod \textsc{bs} = 0 \\ \textsc{bit}((\textsc{bs} \dot{-} 1) \dot{-} (|u| \bmod \textsc{bs}), a_{\lfloor |u|/\mathrm{BS} \rfloor + 1}) & \text{else.} \end{cases}$$

where $\textsc{bs} = \max\{2^{||a_i||} : 1 \leq i \leq |x|\}$, and define

$$\begin{aligned} h(0) &= 0 \\ h(s_i(u)) &= s_{k(u)}(h(u)). \end{aligned}$$

Let

$$z = \tau(t, |x|)$$

where $t = h(N)$ and $|N| = |x| \cdot \textsc{bs}$. Multiplication [resp. division] by power of 2 is possible in A_0 by using the function *pad* [resp. *msp*]. The binary representation of $h(N)$ is of the form

$$\overbrace{\underbrace{10 \cdots 0a_1}_{\mathrm{BS}} 10 \cdots 0a_2 1 \cdots\cdots 0a_{|x|}}^{|x| \cdot \mathrm{BS}}$$

Finally define the function $\beta(i, z)$ to extract the i-th element

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

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

$$\beta(i, z) = \textsc{lsp}\left(\textsc{msp}\left(\pi_1(z), (\ell h(z) \dot{-} i) \cdot \left\lfloor \frac{\pi_1(z)}{\ell h(z)} \right\rfloor\right), \left\lfloor \frac{\pi_1(z)}{\ell h(z)} \right\rfloor \dot{-} 1\right).$$

■

Remark 3.16 The class A_0 has some equivalent classes, such as $\textsc{ac}^0$, $\mathcal{F}\textsc{lh}$, CRAM[1] which is definable by *concurrent parallel random access machine* computing in constant time, and FO definable functions.[2] Each equations were proofed as following,

- FO =CRAM[1] (Immerman [11]),
- $\textsc{ac}^0$ =CRAM[1] (Stockmeyer and Vishkin [17]),
- $FO = \mathcal{F}\textsc{lh}$ (Barrington, Immerman and Straubing [1]),
- $FO = A_0$ (Clote [4]), and
- $\mathcal{F}\textsc{lh}$= A_0 (Clote [6]).

[2] See [1, 11] for definition of FO.

3.2 Polynomial time computable functions

In [8] A.Cobham first isolate the machine independent characterization of polynomial time computable functions as using *bounded recursion on notation.*[3] Although his characterization has yielded a number of applications, unsatisfying aspect of his characterization arises in the bounded recursion on notation. The recursion operator is a powerful one which, however, can only be applied when an explicit size bound is satisfied by the resulting function. Additionally, an initial function $2^{|x|\cdot|y|}$ is needed solely to provide a large enough bound for making recursive definitions.

Recently, certain *unbounded* recursion schemes have been introduced to characterize polynomial time functions, such as Leivant [13] introduced ramified recurrence, Immerman [10] and others have characterized polynomial time computable relations in a way which is also resource free in the sense that there are no explicit bounds in the defining expressions. In this paper the author picks up the following two results, *safe recursion* by Bellantoni and Cook [3] and *full concatenation recursion* by Ishihara [12].

3.2.1 Bounded recursion on notation

Definition 3.17 The function f is defined by *bounded recursion on notation* (BRN) from g, h_0, h_1, k 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 $x, \vec{y}$.

We define the algebra $[0, I, s_0, s_1, \#; \text{COMP}, \text{ BRN}]$ and name it $\mathcal{L}$. To show that $\mathcal{L}$ has the properties of A_0, the following lemma is needed.

Lemma 3.18

$$A_0 \subseteq [0, I, s_0, s_1, \#; \text{COMP,BRN}]$$

Proof. It suffices to show that the function algebra on the right-hand side contains $|x|$ and BIT, and is closed under CRN. CRN is easily simulated by BRN using $\#$ function to bound. And the function $|x|$ is defined by

$$\begin{aligned} |0| &= 0 \\ |s_i(x)| &= s(|x|), \end{aligned}$$

[3] Ritchie's work [14] of the machine independent characterization of the linear space computable functions was actually prior to that of Cobham.

where

$$\begin{aligned} s(0) &= 1 \\ s(s_0(x)) &= s_1(x),\ \text{if } x \neq 0 \\ s(s_1(x)) &= s_0(s(x)). \end{aligned}$$

$s(x)$ is usual successor function. And let

$$\begin{aligned} \text{MOD2}(0) &= 0 \\ \text{MOD2}(s_0(x)) &= 0 \text{ if } x \neq 0 \\ \text{MOD2}(s_1(x)) &= 1 \end{aligned}$$

$$\begin{aligned} \text{TR}(0) &= 0 \\ \text{TR}(s_i(x)) &= x \end{aligned}$$

$$\begin{aligned} msp(x, 0) &= x \\ msp(s, s_i(y)) &= \text{TR}(msp(x, y)), \end{aligned}$$

where $|x|, \text{MOD2}(x), \text{TR}(x), msp(x, y)$ are bounded by x, and $s(x) \leq x\#x$. We have

$$\text{BIT}(y, x) = \text{MOD2}(msp(x, \text{TR}(Exp(y, x)))),$$

where $Exp(i, x)$ is the *bounded exponential* function, as follow

$$Exp(y, x) = 2^{\min(y,|x|)}.$$

To show that $Exp(y, x)$ is in the class $\mathcal{L}$, define $pred(x) = x \dot{-} 1$ by

$$\begin{aligned} pred(0) &= 0 \\ pred(s_0(x)) &= s_1(pred(x)) \text{ if } x \neq 0 \\ pred(s_1(x)) &= s_0(x), \end{aligned}$$

and $pred(x)$ is bounded by x. Define $y \dot{-} |x|$ by

$$\begin{aligned} y \dot{-} |0| &= y \\ y \dot{-} |s_i(x)| &= pred(y \dot{-} |x|), \end{aligned}$$

let

$$\begin{aligned} cond(0, y, z) &= y \\ cond(s_i(x), y, z) &= z \end{aligned}$$

provided $y \dot{-} |x| \leq y$, $cond(x, y, z) \leq y\#z$. Let $H(x, y, z) = cond(y \dot{-} |x|, z, s_0(z))$ then

$$\begin{aligned} Exp(y, |0|) &= 1 \\ Exp(y, |s_i(x)|) &= H(x, y, Exp(y, |x|)). \end{aligned}$$

■

Therefore one have a recursion theoretic characterization of $\mathcal{F}$PTIME. From previous discussion, Turing machine configurations (TM and RATM) are easily expressed in A_0. Thus by Lemma 3.15 and 3.18 configurations are expressed in $\mathcal{L}$. 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 $n_i \in$**N**. The u_i present 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 tapes 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 $\leq n_i$ [resp. $> n_i$]). Assuming some simple binary encoding of $\Gamma \cup \{B\}$, a RATM configuration can be represented using the tupling function by

$$\tau_{2k+5}(q, l_1, r_1, \ldots, \ell_{k+2}, r_{k+2})$$

Let INITIAL$_M(x)$ be the function mapping x to the initial configuration of RATM M on input x. For configurations α in the computation of RATM M on x, let function NEXT$_M(x, \alpha)$ is next configuration of α on input x.

Theorem 3.19 (A. Cobham [8], see H. Rose [15])

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

Proof. Consider first the inclusion from left to right. It is routine to show that INITIAL$_M$, NEXT$_M$ are definable in $\mathcal{L}$ (even A_0). For instance, a move of the first tape head to the right would mean that in the next configuration $\ell_1' = 2 \cdot \ell_1 + \text{MOD}2(r_1)$ and $r' = \lfloor r_1/2 \rfloor$. By suitably composing $0, s_0, s_1, \#$ there is a function $k \in \mathcal{L}$ satisfying $p(|x|) \leq |k(x)|$ for all inputs x. If $p(|x|) = |x|^c$ then $k(x)$ is the function operating COMP on $\#$ with $c-1$ times. Using BRN, define

$$\begin{aligned} Run_M(x, 0) &= \text{INITIAL}_M(x) \\ Run_M(x, s_i(y)) &= \text{NEXT}_M(x, Run_M(x, y)). \end{aligned}$$

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

The inclusion from right to left is proved by an induction on the derivation of the function in $\mathcal{L}$. $\mathcal{F}$PTIME contains the initial functions of $\mathcal{L}$, and closed under COMP and BRN. ■

3.2.2 Full concatenation recursion on notation

Definition 3.20 Assume that $h_0(x,\vec{y},z), h_1(x,\vec{y},z) \le 1$. The function f is defined by *full concatenation recursion on notation* (FCRN) from g, h_0, h_1 if

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

The scheme FCRN is strong version of CRN. This scheme was first introduced by H. Ishihara [12].

Theorem 3.21 (H. Ishihara [12])

$$\mathcal{F}\text{PTIME} = [0, I, s_0, s_1, \text{BIT}, |x|, \#; \text{COMP}, \text{FCRN}]$$

Proof. By theorem 3.19 we will show that

$$\mathcal{L} = [0, I, s_0, s_1, \text{BIT}, |x|, \#; \text{COMP}, \text{FCRN}]$$

Consider the direction from right to left. According to the lemma 3.18, $\mathcal{L}$ contains BIT, $|x|$. It thus suffices to show that $\mathcal{L}$ is closed under FCRN. Let

$$H_i(x,\vec{y},z) = cond(h_i(x,\vec{y},z), s_0(z), s_1(z)) \quad (i = 0,1).$$

Hence, f is definable by BRN as follows,

$$\begin{aligned} f(0,\vec{y}) &= g(\vec{y}), \\ f(s_0(x),\vec{y}) &= H_0(x,\vec{y},f(x,\vec{y})), \text{ if } x \neq 0 \\ f(s_1(x),\vec{y}) &= H_1(x,\vec{y},f(x,\vec{y})). \end{aligned}$$

This function is bounded by $2^{|g(x)|+|x|} = pad(g(\vec{y})\#1, x)$.

Consider now the direction from left to right. We will show that the function algebra on the right-hand side, say $\mathcal{C}$, is closed under BRN. Since FCRN is strong version of CRN, trivially $\mathcal{C}$ contains A_0, $\mathcal{C}$ has the property of A_0.

Now suppose that a function f is defined by BRN from g, h_0, h_1, k in $\mathcal{C}$. We require the function $m(x,\vec{y})$ which bound $k(x,\vec{y})$ always such that $\forall i \le x[k(i,\vec{y}) \le m(x,\vec{y})]$. Define the characteristic function of $|x| < |y|$ as

$$llt(x,y) = sg(msp(y,x)).$$

Let

$$\begin{aligned} p(0,z,x,\vec{y}) &= 0 \\ p(s_i(w),z,x,\vec{y}) &= s_{llt(z,k(msp(x,w),\vec{y}))}(p(w,z,x,\vec{y})). \end{aligned}$$

Then define

$$P(z, x, \vec{y}) = \forall j \leq |x|[|k(\textsc{msp}(x, j), \vec{y})| \leq |z|]$$

has a characteristic function $\overline{sg}(p(s_1(x), z, x, \vec{y}))$, and hence letting

$$\begin{array}{rcl} p'(0, x, \vec{y}) & = & 0 \\ p'(s_i(w), x, \vec{y}) & = & s_{P(k(msp(x,w),\vec{y}),x,\vec{y})}(p'(w, x, \vec{y})), \end{array}$$

$sg(p'(w, x, \vec{y}))$ is a characteristic function of

$$\exists i < |w| P(k(\textsc{msp}(x, i), \vec{y}), x, \vec{y}),$$

in brief

$$\exists i < |w| \forall j \leq |x|[|k(\textsc{msp}(x, j), \vec{y})| \leq |k(\textsc{msp}(x, i), \vec{y})|].$$

We would like to seek the value assigning the maximum value of k in $|x|$. Let

$$\begin{array}{rcl} l(0, x, \vec{y}) & = & 0 \\ l(s_i(w), x, \vec{y}) & = & s_{sg(p'(w,x,\vec{y}))}(l(w, x, \vec{y})). \end{array}$$

Hence letting

$$m(x, \vec{y}) = s_1(k(\textsc{msp}(x, |x| \dot{-} |l(s_1(x), x, \vec{y})|), \vec{y})),$$

then the following predicate is true,

$$P(\lfloor m(x, \vec{y})/2 \rfloor, x, \vec{y}).$$

To simulate BRN by FCRN, we would like construct the history of computing BRN as binary sequence which each block size is $|m(x, \vec{y})|$. Define G by

$$\begin{array}{rcl} G(0, x, \vec{y}) & = & 1 \\ G(s_i(z), x, \vec{y}) & = & s_{\mathrm{BIT}((|m(x,\vec{y})| \dot{-} 1) \dot{-} |z|, g(\vec{y}))}(G(z, x, \vec{y})). \end{array}$$

Then the binary representation of $G(\lfloor m(x, \vec{y})/2 \rfloor, x, \vec{y})$ is of the form

$$\underbrace{10 \cdots 0 g(\vec{y})}_{|m(x,\vec{y})|}.$$

Let

$$\begin{array}{rcl} u(z, x, \vec{y}) & = & \textsc{msp}(x, |x| \dot{-} \lfloor |z|/|m(x, \vec{y})| \rfloor), \\ j(z, x, \vec{y}) & = & \mathrm{BIT}((|x| \dot{-} \lfloor |z|/|m(x, \vec{y})| \rfloor) \dot{-} 1, x), \end{array}$$

the binary representation of x is following

$$\overbrace{\underbrace{u(z, x, \vec{y})}_{\lfloor |z|/|m| \rfloor} \underbrace{j(z, x, \vec{y}) XX \cdots X}_{|x| \dot{-} \lfloor |z|/|m| \rfloor}}^{|x|},$$

where X is one bit constructing x. Then

$$s_{j(z,x,\vec{y})}(u(z,x,\vec{y}))$$

is the most significant $(\lfloor |z|/|m(x,\vec{y})| \rfloor + 1)$-bits of x and is the concatenation $u(z,x,\vec{y})$ and $j(z,x,\vec{y})$. Let

$$\begin{array}{rcl}
a(z,x,\vec{y},w) & = & \mathrm{LSP}(\mathrm{MSP}(w, |x| \bmod |m(x,\vec{y})|), |m(x,\vec{y})| \dot{-} 1), \\
a'(z,x,\vec{y},w) & = & \begin{cases} h_0(u(z,x,\vec{y}),\vec{y},a(z,x,\vec{y},w)) & \text{if } j(z,x,\vec{y}) = 0, \\ h_1(u(z,x,\vec{y}),\vec{y},a(z,x,\vec{y},w)) & \text{otherwise.} \end{cases} \\
b_0(z,x,\vec{y},w) & = & \mathrm{BIT}((|m(x,\vec{y})| \dot{-} 1) \dot{-} (|z| \bmod |m(x,\vec{y})|), a'(z,x,\vec{y},w)), \\
b(z,x,\vec{y},w) & = & \begin{cases} 1 & \text{if } |z| \bmod |m(x,\vec{y})| = 0, \\ b_0(z,x,\vec{y},w) & \text{otherwise.} \end{cases}
\end{array}$$

then $a(z,x,\vec{y},w)$ is of the form

$$\overbrace{XX\cdots X\,\underbrace{a(z,x,\vec{y},w)}_{|m|\dot{-}1}\;\;\underbrace{XX\cdots X}_{|z| \bmod |m(x,\vec{y})|}\;\;}^{|w|}.$$

Define F by FCRN as follows:

$$\begin{array}{rcl}
F(0,x,\vec{y}) & = & G(\lfloor m(x,\vec{y})/2 \rfloor, x, \vec{y}) \\
F(s_i(z),x,\vec{y}) & = & s_{b(z,x,\vec{y},F(z,x,\vec{y}))}(F(z,x,\vec{y})).
\end{array}$$

Then

$$\begin{array}{rcl}
a(z,x,\vec{y},F(z,x,\vec{y})) & = & f(u(z,x,\vec{y}),\vec{y}), \\
a'(z,x,\vec{y},F(z,x,\vec{y})) & = & f(s_{j(z,x,\vec{y})}(u(z,x,\vec{y})),\vec{y}),
\end{array}$$

and hence the binary representation of $F(z,x,\vec{y})$, which is the history of BRN, is of the form

$$\underbrace{10\cdots g(\vec{y})}_{|m(x,\vec{y})|}\overbrace{\underbrace{10\cdots f(1,\vec{y})}_{|m(x,\vec{y})|}\cdots\underbrace{10\cdots f(u(z,x,\vec{y}),\vec{y})}_{|m(x,\vec{y})|}\underbrace{10\cdots f(s_{j(z,x,\vec{y})}(u(z,x,\vec{y})),\vec{y})}_{|m(x,\vec{y})|}\cdots\underbrace{10\cdots f(\lceil |z|/|m| \rceil,\vec{y})}_{|z| \bmod |m(x,\vec{y})|}}^{|z|}$$

Therefore

$$f(x,\vec{y}) = \mathrm{LSP}(F(\lfloor x \# m(x,\vec{y})/2 \rfloor, x, \vec{y}), |m(x,\vec{y})| \dot{-} 1),$$

and so f is definable in $\mathcal{C}$. Thus $\mathcal{C}$ closed under BRN. ■

Using same recursion, we introduce the machine independent characterization of the polynomial time hierarchy.

Definition 3.22 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

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

Theorem 3.23 (Folklore)

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

Corollary 3.24

$$\mathcal{F}\text{PH} = [0, I, s_0, s_1, \text{BIT}, |x|, \#; \text{COMP}, \text{FCRN}, \text{BMIN}].$$

Proof. By theorem 3.21 and 3.23, it is trivial. ■

3.2.3 Safe recursion on notation

Bellantoni and Cook [3] introduced not using smash function $2^{|x| \cdot |y|}$ and certain unbounded recursion schemes 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_i 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. If $\mathcal{F}$ and $\mathcal{O}$ are collections of initial functions and operations which distinguish normal and safe variables, then 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, (NORMAL$\cap[\mathcal{F}; \mathcal{O}])_*$ denotes the collection of predicates whose characteristic function $f(\vec{x};)$ has only normal variables and belongs to $[\mathcal{F}; \mathcal{O}]$. Bellantoni and Cook showed the machine independent characterization of the class of polynomial time computable functions using this notion.

Definition 3.25 Define the following initial functions by

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

Definition 3.26 (Bellantoni-Cook [3]) 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 3.27 The function f is defined by *safe recursion on notation*[4] (SRN) from the functions g, h_0, h_1 if

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

The function algebra B is defined by

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

Example 3.28 We can define the function $Smash(y, x;) = 2^{|x| \cdot |y|}$ in B. Let

$$\begin{array}{rcl} Pad(0; y) & = & y \\ Pad(s_i(x); y) & = & S_1(; Pad(x; y)) \end{array}$$

Define

$$\begin{array}{rcl} Smash(0, x;) & = & 1 \\ Smash(s_i(y), x;) & = & Pad(x; Smash(y, x;)) \end{array}$$

Theorem 3.29 (Bellantoni-Cook [3]) *The polynomial time computable functions are exactly those functions of B having only normal arguments, i.e.*

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

Let $\square_i^P$ 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, $\mathcal{F}$PH$= \cup_i \square_i^P$.

Theorem 3.30 (S.Bellantoni [2]) *Let μFP_i be the Cobham class augmented by allowing i applications of* BMIN*. Then for $i \geq 0$,*

$$\square_i^P = \mu FP_i.$$

[4] In [3] this scheme is called *predicative recursion on notation.*

Definition 3.31 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 $\mu B = [0, I, S_0, S_1, P, C;$ SCOMP,SRN,SMIN$]$. Let μB_i denote the set of functions derivable in μB using at most i applications of safe minimization.

Theorem 3.32 (S.Bellantoni [2])

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

Remark 3.33 Bellantni characterized the other complexity classes by same notion; LOGSPACE, NC and the class of linear space computable functions. Clote also has given the characterization of ETIME functions by linear growth. S. Bloch characterized ALOGTIME by this notion.

Chapter 4

Constable's Class $\mathcal{K}$

4.1 The class $\mathcal{K}$ and some parallel complexity classes

R. Constable [9] defined the class $\mathcal{K}$ drawing on polynomial analogy with the class of Kalmár elementary functions $\mathcal{E}$. Since $\mathcal{E}$ embodies exponential function, we can not regard $\mathcal{E}$ as feasible class. And Clote [5] showed the relation $\mathcal{K}$ and some parallel complexity classes.

Definition 4.1 The function f is defined by *bounded summation* (BSUM) [resp. *bounded product* (BPROD)][1] from g if $f(x, \vec{y})$ equals

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

Definition 4.2 (R. Constable [9]) The function f is defined by *sharply bounded summation* (SBSUM) [resp. *sharply bounded product* (SBPROD)][2] from g if $f(x, \vec{y})$ equals

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

The elementary functions were first introduced by Kalmár.

Definition 4.3 The class $\mathcal{E}$ of *elementary* functions is the algebra

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

Definition 4.4 We definethe class $\mathcal{K}$ by

$$[0, I, s_0, s_1, +, \dot{-}, \times, \lfloor x/y \rfloor; \text{COMP,SBSUM,SBPROD}].$$

[1] Bounded summation [resp. product] is sometimes called *limited summation* [resp. *product*].

[2] Sharply bounded summation [resp. product] is sometimes called *weak summation* [resp. *weak product*].

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

Definition 4.5 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 by BRN and $f(x, \vec{y}) = F(|x|, \vec{y})$; i.e.

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

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

Though one needs $|x|$ times recursion to define $f(x)$ by BRN, one requires $||x||$ times recursion in the case of WBRN.

Theorem 4.6 (Clote [4]) *Let the algebra*

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

then

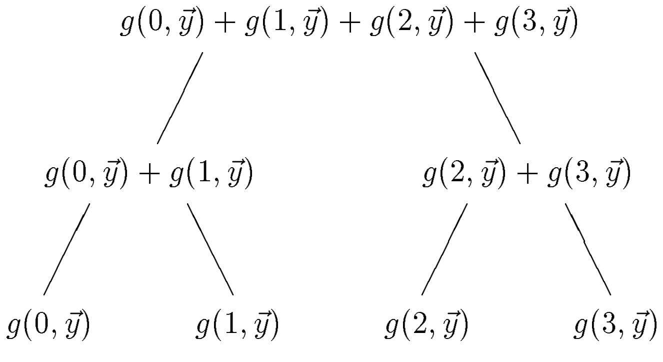
$$\text{NC} = A.$$

Theorem 4.7 (Clote-Takeuti [7])

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

Lemma 4.8 *A is closed under* SBSUM.

Proof. Let $f(x, \vec{y}) = \sum_{i \leq |x|} g(i, \vec{y})$, where $g \in A$. We must show that $f(x, \vec{y}) \in A$. Since if we added g one by one, $|x|$ steps would be needed. Make binary tree where values $g(i, \vec{y})$ are placed at the leaves and internal nodes have as values the sum of their two children. Addition is executed the number of nodes. And for instance $|x| = 3$, binary tree is given as follows.


$g(0, \vec{y}) + g(1, \vec{y}) + g(2, \vec{y}) + g(3, \vec{y})$
$g(0, \vec{y}) + g(1, \vec{y})$
$g(2, \vec{y}) + g(3, \vec{y})$
$g(0, \vec{y})$
$g(1, \vec{y})$
$g(2, \vec{y})$
$g(3, \vec{y})$

By previous chapter, the algebra A_0 contains the functions $+, \dot{-}, rev$. Clearly $A_0 \subseteq A$. We show the *bounded exponential* function is in this class. The bounded exponential function is

$$Exp(x, y) = 2^{\min(x,|y|)},$$

to use $Exp(x, y)$ as 2^x, if x is bounded by $|y|$. Firstly define the auxiliary function $exp0$

$$\begin{aligned} exp0(x, y, 0) &= 1 \\ exp0(x, y, s_i(z)) &= \begin{cases} s_1(exp0(x, y, z)) & \text{if } |z| = x \le |y| \vee |z| = |y| \le x \\ s_0(exp0(x, y, x)) & \text{else.} \end{cases} \end{aligned}$$

Then

$$Exp(x, y) = \lfloor rev(exp0(x, y, s_1(y)))/2 \rfloor .$$

Define functions $\ell(x)$ [resp. $n(x)$] whose value is the number of *leaves* [resp. *nodes*] of a full binary tree T_x which will be associated with x;

$$\begin{aligned} \ell(x) &= Exp(||x||, 2 \cdot |x|) \\ n(x) &= 2 \cdot \ell(x) - 1. \end{aligned}$$

While

$$n(x) = 2^{||x||+1} - 1 < 4 \cdot |x| < |2^{4 \cdot |x|}| = |8\#x|.$$

As $Exp(n(x), 8\#x) = 2^{n(x)}$, the function $2^{n(x)}$ is definable in A_0 and A.

Let T_x be the full binary tree having $\ell(x)$ many leaves and altogether $n(x)$ many nodes, whose nodes labeled as follows: leaves are labeled $0, \ldots, \ell(x) - 1$ form left to right, then parents of leaves are labeled $\ell(x), \ldots, \ell(x) + \frac{\ell(x)}{2} - 1$ form left to right, etc.

We will explain more formally. Let

$$\begin{aligned} LC(i, x) &= n(x) - [2 \cdot (n(x) - i) + 1] \\ RC(i, x) &= n(x) - [2 \cdot (n(x) - i)]. \end{aligned}$$

Hence $LC(i, x)$ [resp. $RC(i, x)$] is the labeled number of left [resp. right] child of i in T_x, provided that $\ell(x) \le i < n(x)$.

By this idea we will formalize a computation on the tree. T_x. Define

$$f(i, x, \vec{y}) = \begin{cases} g(i, \vec{y}) & \text{if } i \le |x| \\ 0 & \text{if } |x| < i < \ell(x) \\ f(LC(i, x), x, \vec{y}) + f(RC(i, x), x, \vec{y}) & \text{if } \ell(x) \le i < n(x) \\ 0 & \text{if } n(x) \le i \end{cases}$$

It is easily verified that

$$f(n(x) - 1, x, \vec{y}) = \sum_{i=0}^{|x|} g(i, \vec{y})$$

since f assigns values $g(0, \vec{y}), \ldots, g(|x|, \vec{y})$ to the leaves of T_x and then computes level-by-level the pairwise sums so that $\sum_{i=0}^{|x|} g(i, \vec{y})$ is assigned at the root $n(x) - 1$ of T_x.

By the lemma 3.15, functions to encode the sequence were shown to belong to A_0. It is then straightforward to show that a sequence concatenation function $\frown$ exists in A_0 for which if $s = \langle s_1, \ldots, s_n\rangle$ and $t = \langle t_1, \ldots, t_m\rangle$ then $s^\frown t = \langle s_1, \ldots, s_n, t_1, \ldots, t_m\rangle$. By $\pi_2(s) = n, \pi_2(t) = m$,

$$s^\frown t = \langle \beta(1, s), \ldots, \beta(\pi_2(s), s), \beta(1, t), \ldots, \beta(\pi_2(t), t)\rangle.$$

The reason of using encoding function is that we would like to refer to the earlier history of sum than just before node.

Now define $F(z, x, \vec{y})$ by

$$\begin{aligned} F(0, x, \vec{y}) &= \langle\ \rangle \\ f(s_i(z), x, \vec{y}) &= F(z, x, \vec{y})^\frown\langle h(z, x, \vec{y})\rangle \end{aligned}$$

where

$$h(z, x, \vec{y}) = \begin{cases} g(|z|, \vec{y}) & \text{if } |z| < |x| \\ 0 & \text{if } |x| \le |z| < \ell(x) \\ \beta(F(z, x, \vec{y}), LC(|z|, x)) + \beta(F(z, x, \vec{y}), RC(|z|, x)) & \text{if } \ell(x) \le |z| < n(x) \\ 0 & \text{else.} \end{cases}$$

F become the history of addition. Thus $F(z, x, \vec{y}) = \langle f(0, x, \vec{y}), \ldots, f(|z| - 1, x, \vec{y})\rangle$. As to have seen $f(n(x) - 1, x, \vec{y}) = \sum_{i=0}^{|x|} g(i, \vec{y})$. In the discussion of chapter 2 it was shown that if $g \in A_0$, then the *maximum* function

$$m_g(x, \vec{y}) = \max\{g(i, \vec{y}) : i \le |x|\}$$

belongs to A_0. It follows that if $g \in A$, then $m_g \in A$. Hence,

$$\begin{aligned} \sum_{i=0}^{|x|} g(i, \vec{y}) &\le m_g(x, \vec{y}) \cdot (|x| + 1) \\ &\le m_g(x, \vec{y}) \# (2 \cdot x) \end{aligned}$$

and so $\sum_{i=0}^{|x|} g(i, \vec{y})$ is bounded by a function in A. Hence function F is also bounded in A.

Let $G(z, x, \vec{y}) = F(|z|, x, \vec{y})$. Then G is definable in A by using WBRN. Then A contains G. Finally, define K by

$$\begin{aligned} K(x, \vec{y}) &= G(2^{n(x)} \dot{-} 1, x, \vec{y}) \\ &= F(|2^{n(x)} \dot{-} 1|, x, \vec{y}) \\ &= F(n(x), x, \vec{y}) \\ &= \langle f(0, x, \vec{y}), \ldots, f(n(x) - 1, x, \vec{y})\rangle. \end{aligned}$$

Thus

$$\begin{aligned} \beta(K(x, \vec{y}), n(x)) &= f(n(x) - 1, x, \vec{y}) \\ &= \sum_{i=0}^{|x|} g(i, \vec{y}) \end{aligned}$$

and hence A is closed under SBSUM. ■

Theorem 4.9 (P. Clote [5]) *$\mathcal{K} \subseteq A$.*

Proof. By replacing sum by product, the proof of the proof of the previous lemma can immediately be modified to yield that A is closed under SBPROD.[3] The initial functions $0, I, s_0, s_1, +, \times$ of $\mathcal{K}$ all belong to A. The first three belong to A by definition; in previous chapter, it was shown that $+ \in A_0 \subseteq A$; $\times$ is known to belong to NC (even NC^1). The class A is closed under composition, and by previous lemma, under SBSUM and SBPROD. It follows that $\mathcal{K} \subseteq A$. ■

Theorem 4.10 (P. Clote [5]) *TC$^0 \subseteq \mathcal{K}$.*

Proof. Initially it is necessary that the initial functions of TC^0 can be defined in $\mathcal{K}$.

$$\begin{aligned}
|x| &= \left(\sum_{i\leq|x|} 1\right) \dot{-} 1 \\
2^{|x|} &= \left\lfloor \frac{\prod_{i\leq|x|} 2}{2} \right\rfloor \\
\overline{sg}(x) &= 1 \dot{-} x = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{else} \end{cases} \\
sg(x) &= 1 \dot{-} \overline{sg}(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 & \text{else} \end{cases} \\
c_{\leq}(x,y) &= sg(x \dot{-} y) = \begin{cases} 1 & \text{if } x \leq y \\ 0 & \text{else} \end{cases} \\
cond(x,y,z) &= \overline{sg}(x)\cdot y + sg(x)\cdot z = \begin{cases} y & \text{if } x = 0 \\ z & \text{else} \end{cases} \\
\min(x,y) &= cond(x \dot{-} y, x, y).
\end{aligned}$$

From these functions, definitions by case are admissible in $\mathcal{K}$. Definitions of the form $f(x,\vec{y}) = \sum_{i\leq|x|} h(i,x,\vec{y})$ and $f(x,\vec{y}) = \prod_{i\leq x} h(i,x,\vec{y})$ are allowed by SBSUM and SBPROD. Define

$$Exp(x,y) = 2^{\min(x,|y|)} = \prod_{i\leq|y|} g(i,x,y)$$

[3] In showing that G is definable by SBPROD, note that

$$\prod_{i=0}^{|x|} g(i,\vec{y}) \leq m_g(x,\vec{y})^{|x|+1} \leq m_g(x,\vec{y})\#(2\cdot x).$$

where

$$g(i,x,y)=\begin{cases} 2 & \text{if } i<\min(x,|y|) \\ 1 & \text{else} \end{cases}$$

Further define

$$\begin{aligned} Msp(x,i) &= \left\lfloor \frac{x}{Exp(i,x)} \right\rfloor \\ \textsc{bit}(i,x) &= Msp(x,i) \dot{-} 2 \cdot \left\lfloor \frac{Msp(x,i)}{2} \right\rfloor \\ x\#y &= \left\lfloor \frac{\prod_{i\le|y|} 2^{|y|}}{2^{|x|}} \right\rfloor = 2^{|x|\cdot|y|} \end{aligned}$$

Thus all the initial functions of $\textsc{tc}^0$ belong to $\mathcal{K}$.

Secondly we show that $\mathcal{K}$ is closed under $\textsc{crn}$, suppose that f is defined from $g, h_0, h_1 \in \mathcal{K}$ using $\textsc{crn}$. To simulate $\textsc{crn}$ by $\textsc{sbsum}$, consider the summation such that

$$h_i(z,\vec{y}) \cdot 2^{|x|-1-z}, \quad 0 \le z \le |x|.$$

We define formally, let

$$U(z,x,\vec{y}) = h_{\mathrm{BIT}(z,x)}(Msp(x,z+1),\vec{y}) \cdot Msp(2^{|x|\dot{-}1}, |x| \dot{-} (z+1)).$$

Hence f is definable as follow using $\textsc{sbsum}$,

$$f(x,\vec{y}) = \sum_{i\le|x|} U(i,x,\vec{y}) + g(\vec{y}) \cdot 2^{|x|}.$$

Thus $\mathcal{K}$ contains all the initial functions of $\textsc{tc}^0$ and is closed under the operations $\textsc{comp}$ and $\textsc{crn}$, hence $\textsc{tc}^0 \subseteq \mathcal{K}$. ■

4.2 Some observations on $\mathcal{K}$

The class $\mathcal{K}$ is very natural, however the corresponding machine model is unknown. Then following question has been proposed.

Question 4.11 What complexity class corresponds to

$$[0, I, s_0, s_1, +, \dot{-}, \times, \lfloor x/y \rfloor; \textsc{comp}, \textsc{sbsum}, \textsc{sbprod}]?$$

According to [5, 6], H.-J. Burtshick has proposed that polynomial size uniform arithmetic circuits could be related to the $\mathcal{K}$. H. Ishihara suggested that the algebra adding initial function $x^{|y|}$ to $\textsc{tc}^0$ could be involving in $\mathcal{K}$. And Ishihara showed that $\textsc{tc}^0$ is closed under $\textsc{sbsum}$.

Lemma 4.12 $\textsc{tc}^0$ *is closed under* $\textsc{sbsum}$.

Proof. Let

$$m(x,\vec{y}) = \max\{g(i,\vec{y}) : i \le |x|\}\#(2\cdot x)$$

and

$$f(x,\vec{y}) = \textsc{bit}((2^{||m(x,\vec{y})||} \mathbin{\dot{-}} 1) \mathbin{\dot{-}} (|x| \bmod 2^{||m(x,\vec{y})||}), g(\lfloor |x|/2^{||m(x,\vec{y})||}\rfloor, \vec{y})).$$

Then define h and h' by $\textsc{crn}$ as follow:

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

$$\begin{aligned} h'(0,\vec{y}) &= 0 \\ h'(s_i(x),\vec{y}) &= s_{\overline{sg}(|x| \bmod 2^{||m(x,\vec{y})||})}(h'(x,\vec{y})). \end{aligned}$$

We would like to show the existence of the function $t(\in \textsc{tc}^0)$ such that

$$|t(x,\vec{y})| = 2^{||m(x,\vec{y})||}\cdot(||x||+1).$$

Let

$$t_1(x,\vec{y}) = (|m(x,\vec{y})|\#1)(|x|+1),$$

and

$$\begin{aligned} t_2(x,\vec{y},0) &= 1 \\ t_2(x,\vec{y},s_i(z)) &= \begin{cases} s_0(t_2(x,\vec{y},z)) & \text{if } |z| \le t_1(x,\vec{y}) \\ s_1(t_2(x,\vec{y},z)) & \text{else .} \end{cases} \end{aligned}$$

For z such that $|z| \ge t_1(x,\vec{y})$, it is clear that binary representation of $t_2(x,\vec{y},z)$ is

$$1\overbrace{\underbrace{00\cdots0}_{t_1(z,\vec{y})}11\cdots1}^{|z|}$$

where there are $t_1(x,\vec{y})$ many 0's and $|z| - t_1(x,\vec{y},z)$ many 1's.

Now letting,

$$t_3(x,\vec{y}) = ((m(x,\vec{y}) * m(x,\vec{y}))\#x)\cdot((m(x,\vec{y}) * m(x,\vec{y}))\#1),$$

clearly $|t_3(x,\vec{y})| \ge t_1$. Let

$$t_4(x,\vec{y}) = t_2(x,\vec{y},t_3(x,\vec{y}))$$

and

$$t_5(x,\vec{y}) = \mu i < |t_4(x,\vec{y})|[\textsc{bit}(t_4(x,\vec{y}),i) = 0].$$

By the lemma 3.12, t_5 is in $\in\mathrm{TC}^0(\supseteq A_0)$. The binary representation of $t_4(x, \vec{y})$ is

$$1\overbrace{\underbrace{00\cdots0}_{2^{||x||}\cdot(|x|+1)}\ 11\cdots1}^{|t_3(x,\vec{y})|}$$

where there are $2^{||x||} \cdot (|x| + 1)$ many 0's and $|t_3(x, \vec{y})| - 2^{||x||} \cdot (|x| + 1)$ many 1's. As well, $t_5(x, \vec{y})$ is the position of the rightmost 0 occurring the binary representation of $t_4(x, \vec{y})$. Now define the desired function t by

$$t(x, \vec{y}) = \mathrm{MSP}(t_4(x, \vec{y}), s_1(t_5(x, \vec{y}))).$$

Putting

$$\begin{array}{rcl} A(x, \vec{y}) & = & h(t(x, \vec{y}), \vec{y}), \\ B(x, \vec{y}) & = & h'(t(x, \vec{y}), \vec{y}). \end{array}$$

the binary representations $A(x, \vec{y})$ and $B(x, \vec{y})$ are of the forms

$$g(0, \vec{y})\underbrace{0\cdots0g(1, \vec{y})}_{2^{||m(x,\vec{y})||}}\underbrace{0\cdots0g(2, \vec{y})}_{2^{||m(x,\vec{y})||}}\cdots\underbrace{0\cdots0g(|x|, \vec{y})}_{2^{||m(x,\vec{y})||}}$$

and

$$1\underbrace{0\cdots01}_{2^{||m(x,\vec{y})||}}\underbrace{0\cdots01}_{2^{||m(x,\vec{y})||}}\cdots\underbrace{0\cdots01}_{2^{||m(x,\vec{y})||}}$$

respectively, where each blocksize is $2^{||m||}$. The $|x|$-th block of the product $A(x, \vec{y}) \cdot B(x, \vec{y})$ the binary representation of the integer

$$\mathrm{MSP}(\mathrm{LSP}(A(x, \vec{y}) \cdot B(x, \vec{y}), 2^{||m(x,\vec{y})||} \cdot (|x| + 1)), 2^{||m(x,\vec{y})||} \cdot |x|)$$

which is equal to $\sum_{i\leq|x|} g(i, \vec{y})$. ■

If we append initial function $x^{|y|}$ to TC^0 then smash function is expressed by $x\#y = (2^{|x|})^{|y|}$, moreover add $\lfloor x/y \rfloor$.

Definition 4.13 Define the following class.

$$\mathcal{T} = [0, I, s_0, s_1, |x|, \mathrm{BIT}, \times, \lfloor x/y \rfloor, x^{|y|}; \mathrm{COMP}, \mathrm{CRN}].$$

The author attempted to show the equivalence $\mathcal{T} = \mathcal{K}$ to find out any clues to solve question 4.11.

Lemma 4.14 *Let $\binom{x}{y}$ be binary coefficient for $x \geq y > 0$. Then $\binom{|x|}{y}$ is definable in $\mathcal{T}$.*

Proof. Let

$$C(x) = (2^{|x|}+1)^{|x|}.$$

The binary representation of $C(x)$ is

$$\binom{|x|}{0}\underbrace{00\cdots0\binom{|x|}{1}}_{|2^{|x|}|}0\cdots00\cdots0\binom{|x|}{|x|-1}00\cdots0\binom{|x|}{|x|},$$

where each blocksize is $|2^{|x|}|$. Hence for $|x| \geq y \geq 0$,

$$\binom{|x|}{y} = \textsc{msp}(\textsc{lsp}(C(x),(y+1)\cdot|2^{|x|}|),y\cdot|2^{|x|}|).$$

■

Lemma 4.15

$$n! = \sum_{i=0}^{n}\left((-1)^i\cdot(n-i)^n\cdot\binom{x}{i}\right).$$

Proof. Choose i elements in $x_1,\ldots,x_n$ then letting y_i as the following way

$$\begin{aligned}
y_n &= (x_1+x_2+\cdots+x_n)^n\\
y_{n-1} &= (x_1+\cdots+x_{n-1})^n+(x_1+x_3+\cdots+x_n)^n+\cdots+(x_1+\cdots+x_{n-1})^n+(x_2+\cdots+x_n)^n\\
y_{n-2} &= (x_1+\cdots+x_{n-2})^n+(x_1+\cdots+x_{n-3}+x_n)^n+\cdots+(x_3+\cdots+x_n)^n\\
&\vdots\\
y_2 &= (x_1+x_2)^n+(x_2+x_3)^n+\cdots+(x_{n-1}+x_n)^n\\
y_1 &= x_1^n+x_2^n+\cdots+x_n^n.
\end{aligned}$$

Each y_i has the sum of the $\binom{n}{i}$ terms which is the sum of i variables to the n-th power.

And let z_i in the ensuing

$$z_i = \sum_{j_1\neq j_2\neq\cdots\neq j_i,\, m_1+m_2+\cdots+m_i=n}\binom{n}{m_1,m_2,\ldots,m_i}x_{j_1}^{m_1}x_{j_2}^{m_2}\cdots x_{j_i}^{m_i},$$

where $\binom{n}{m_1,m_2,\ldots,m_i}$ is multinomial coefficient such that

$$\binom{n}{m_1,m_2,\ldots,m_i} = \frac{n!}{m_1!m_2!\cdots m_i!},$$

for instance if $n = 3$ then

$$\begin{aligned}
z_1 &= \binom{3}{3}x_1^3 + \binom{3}{3}x_2^3 + \binom{3}{3}x_3^3\\
z_2 &= \binom{3}{1,2}x_1x_2^2 + \binom{3}{1,2}x_1x_3^2 + \binom{3}{1,2}x_2x_3^2 + \binom{3}{2,1}x_1^2x_2 + \binom{3}{2,1}x_1^2x_3 + \binom{3}{2,1}x_2^2x_3\\
z_3 &= \binom{3}{1,1,1}x_1x_2x_3,
\end{aligned}$$

note that $\binom{3}{1,1,1} = 3!$. Therefore

$$\begin{aligned}
y_n &= z_1 + z_2 + \cdots + z_n\\
y_{n-1} &= \binom{n-1}{1}z_1 + \binom{n-2}{1}z_2 + \cdots + \binom{1}{1}z_{n-1}\\
y_{n-2} &= \binom{n-1}{2}z_1 + \binom{n-2}{2}z_2 + \cdots + \binom{2}{2}z_{n-2}\\
&\vdots\\
y_i &= \binom{n-1}{n-i}z_1 + \binom{n-2}{n-i}z_2 + \cdots + \binom{n-i}{n-i}z_i\\
&\vdots\\
y_2 &= \binom{n-1}{n-2}z_1 + \binom{n-2}{n-2}z_2\\
y_1 &= \binom{n-1}{n-1}z_1.
\end{aligned}$$

By

$$\sum_{i=0}^{n}\left((-1)^i \cdot \binom{n}{i}\right) = 0,$$

thus

$$\begin{aligned}
\sum_{i=1}^{n}\left((-1)^{i-1} \cdot y_i\right) &= z_n\\
&= n! \cdot x_1x_2\cdots x_n.
\end{aligned}$$

On the other hand if we suppose that all x_i is 1 then

$$\begin{aligned}
y_n &= n^n \cdot \binom{n}{n}\\
y_{n-1} &= (n-1)^n \cdot \binom{n}{n-1}\\
&\vdots
\end{aligned}$$

$$\begin{aligned} y_i &= (n-i)^n \cdot \binom{n}{i} \\ &\vdots \\ y_2 &= 2^n \cdot \binom{n}{2} \\ y_1 &= 1^n \cdot \binom{n}{1}, \end{aligned}$$

Hence this lemma is proofed. ■

Lemma 4.16 *$\mathcal{T}$ contains the factorial of the length of x; $|x|!$.*

Proof. By the lemma 4.12, 4.15 and $\textsc{tc}^0 \subseteq \mathcal{T}$,

$$\begin{aligned} |x|! &= \sum_{i=0}^{|x|} \left((-1)^i \cdot (|x|-i)^{|x|} \cdot \binom{|x|}{i} \right) \\ &= |x|^{|x|} - (|x|-1)^{|x|} \cdot \binom{|x|}{|x|-1} + (|x|-2)^{|x|} \cdot \binom{|x|}{|x|-2} - \cdots \mp 2^{|x|} \cdot \binom{|x|}{2} \pm 1^{|x|} \cdot \binom{|x|}{1} \\ &= \sum_{i=0}^{|x|} \left((|x| \dot{-} s_0(i)) \cdot \binom{|x|}{|x| \dot{-} s_0(i)} \right) \dot{-} \sum_{j=0}^{|x|} \left((|x| \dot{-} s_1(j)) \cdot \binom{|x|}{|x| \dot{-} s_1(j)} \right). \end{aligned}$$

■

Remark 4.17 After all we could not show that $\mathcal{T}$ is closed under SBPROD. If one proved it, $\mathcal{T}$ and $\mathcal{K}$ would be equivalent algebra. Therefore the author could not answer the question 4.11. However we have that $\mathcal{T} \subseteq \mathcal{K}$.

Chapter 5

Concluding Remarks

In chapter 2 we have seen the classes which several machine models involve. Turing machine is used regularly as computation model. Typical open problem in structural complexity theory is "Are P and NP properly different class?". Then both of P and NP are originally defined by Turing machine. Circuit families have the advantage of surveying the inside of P.

The author has searched for the class of feasibly computable functions. It is maybe important that the notion of feasibly computable needs both of machine depend and independent characterization. In chapter 3 the class of polynomial time computable functions is mainly discussed. Cobham [8] first finded out a machine independent characterization of polynomial time functions using a certain variant of primitive recursion. And the author also discussed other two recursions. In chapter 4 the Constable's class $\mathcal{K}$ is treated as the class of feasibly computable functions. $\mathcal{K}$ is the smallest class of functions containing the initial functions; constant 0, projections, binary successors $+, \dot{-}, \times, \lfloor x/y \rfloor$ and closed under the operations of composition, sharply bounded summation and sharply bounded product. In [5] Clote obtained the relation with some classes defined by circuit families i.e. $\mathrm{TC}^0 \subseteq \mathcal{K} \subseteq \mathrm{NC}$. However we can not answer the machine dependent characterization of $\mathcal{K}$. The author have attempted to show that $\mathcal{K}$ is equivalent to a function algebraic class related to the complexity class. Then the author define the class $\mathcal{T}$ which is the extension of TC^0. It is clearly that $\mathcal{T} \subseteq \mathcal{K}$. Hence we would like to know that $\mathcal{T}$ contains all initial functions of $\mathcal{K}$ and closed under SBSUM and SBPROD. Then in this paper it is shown that $\mathcal{T}$ contains all initial functions of $\mathcal{K}$ and closed only under SBSUM. Thus the following problem remained,

Question 5.1 Is $\mathcal{T}$ closed under SBPROD ?

The author only showed that $\mathcal{T}$ contains particular binary coefficient $\binom{|x|}{y}$ and also particular factorial function of binary length $|x|!$. Unfortunately the author is coming to reluctant conclusion that question 5.1 remains as still open problem. It is left as a future work.

Acknowledgments

I am very grateful to Associate professor Hajime Ishihara, Professor Hiroakira Ono for their helpful guidance, and kind encouragement and giving me a chance of this work. I would like to thank a member of people in OIL for their friendship.

Bibliography

[1] D. Mix Barrington, N. Immerman, and H. Straubing. On uniform in NC^1. *Journal of Computer and System Science,* 41(3):274-306, 1990.

[2] S. Bellantoni. Predicative recursion and the polytime hierarchy. In P. Clote and J. Remmel, editors, *Feasible Mathematics II,* pages 15-29, Birkhäuser, 1995.

[3] S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity,* 2:97-110, 1992.

[4] P. 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.

[5] 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. Result presented at CSL in Paderborn, 1995. To appear.

[6] P. Clote. Computation Models and Function Algebras. In E. Griffor, editor, *Handbook of Recursion Theory.* in preparation.

[7] 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, 1995.

[8] 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.

[9] R. Constable. Type 2 computational complexity. In *5th Annual ACM Symposium on Theory of Computing,* pp. 108-121, 1973.

[10] N. Immerman. Languages that capture complexity classes. *SIAM Journal of Computing,* 16:760-778, 1987.

[11] N. Immerman. Expressibility and parallel complexity. *SIAM Journal of Computing,* 18:625-638, 1989.

[12] H. Ishihara. Function algebraic characterizations of the polytime functions. *Computational Complexity,* to appear, 1998.

[13] D. Leivant. Stratified functional programs and computational complexity. In *Conference Record of the Twentieth Annual ACM Symposium on Principle of Programing Languages,* 325-333, 1993.

[14] R. W. Ritchie. Classes of predictably computable functions. *Tras. Am. Math. Soc.,* 106:139-173, 1963.

[15] H. E. Rose. *Subrecursion: Function and Hierarchies,* volume 9 of *Oxford Logic Guides.* Clarendon Press, Oxford, 1984. 191 pages.

[16] W. L. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Science,* 22:365-383, 1981.

[17] Y. Stockmeyer and U. Vishkin. Simulation of parallel random access machines by circuits. *SIAM Journal of Computing,* 13:409-422, 1984.