National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Canada

SET LOGIC FOUNDATION OF CARRIER COMPUTING

by

Alioune Ngom

A THESIS
submitted to the School of Graduate Studies and Research
in partial fulfillment of the requirement
for the degree of

Master
in
Computer Science

Ottawa-Carleton Institute of Computer Science
Department of Computer Science
Faculty of Science
University of Ottawa
OTTAWA, ONTARIO

ISBN 0-612-04913-2

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Dr. Ivan Stojmenovic, and to my co-supervisor, Dr. Corina Reischer. Since the undertaking of my thesis, they have been a constant source of support, guidance and ideas for me. I am most appreciative of their patient assistance.

I would also like to thank my Mom and Dad, my sisters and brothers for their constant moral support and encouragement. This would have been a daunting task if they had not been there for me.

Thanks to God for all.

SET LOGIC FOUNDATION OF CARRIER COMPUTING

SET LOGIC FOUNDATION OF CARRIER COMPUTING

ABSTRACT

Set logic algebra (SLA) is a special case of multiple-valued logic algebra. As an ultra higher-valued logic system, a set-valued logic (SVL) system offers a potential and an essential solution to the interconnection problems that occur in highly parallel VLSI systems. The fundamental concept inherent to a SVL system is *multiplex computing* or logic values multiplexing: which means the simultaneous transmission of logic values. This basic concept enables the realization of *superchips* free from interconnection problems. Parallel processing with *multiplexable information carriers* makes possible to construct large-scale highly parallel system with reduced interconnections. Since the multiplexing of logic values increases the information density, several binary functions can be executed in parallel in a single module. Therefore a great reduction of interconnections can be achieved using optimal multiplexing scheme. Possible approaches to the implementation of the SVL system are based on frequencies multiplexing, waves multiplexing and molecules multiplexing, and are called *carrier computing systems*. Our research focuses on the study of completeness properties in SLA under compositions with union ($\cup$), intersection ($\cap$) and complement ($\bar{}$) functions. More precisely, the question is what kind of set logic functions can be constructed from given set of functions which includes $\cup$, $\cap$, and $\bar{}$, i.e. whether any set logic function can be constructed from such set. We classify the set logic functions according to their ability to participate in a base (complete irredundant sets of functions) and describe all bases once the classification is done. We develop also some algorithms (programs) for classification and enumeration of functions and bases, which are very useful for a general completeness analysis.

INTRODUCTION

Motivations

A r-valued set logic is the logic of functions mapping n-tuples of subsets into subsets over r values. Such functions are called set logic functions or set-valued functions. Completeness properties for set logic functions are of great interest to engineers because of recent applications in the design of carrier computing models. This topic presents a special interest for engineers who develop new high-radix logical circuits based on frequency, biological or optical computing, called *carrier computing*. During last several years a number of engineers working in the design of switching circuits became interested in using biological molecular, optical or high-frequency switching elements in order to create simpler and more economical circuits especially suitable for parallel computing. Their research on biological molecular computing, that is computing based on matching of enzymes (the biological catalysts) and their choice of reactants called substrata, suggest the interest of studying set-valued functions and switching devices. Starting from the needs of these designers the class of bio-algebras were introduced as a class of primal algebras, that is capable of model computations done by bio-circuits. Functions implemented by these circuits are defined on the sets of subsets of finite sets and range over similar sets of subsets and, in general, are non Boolean (that is, are not polynomials of the Boolean algebra of all subsets of a set). So, the subsets of the set of Boolean set-valued functions are not complete, that is, they cannot express or produce all the set-valued functions. On the other hand, Boolean set-valued functions (which are a small fraction of the collection of set-valued functions) are very important set-valued functions, because they can be realized using binary standard electronic circuitry. It is shown that r-valued set logic is isomorphic to 2^r-valued logic.

Let U denote the set of all n-place (set logic) functions ($n \geq 1$) ranging over a set V of k values and with all values in V. The set U is called the set of n-ary operations on V in universal algebras and the set of n-place functions of k-valued logic in the propositional calculus of k-valued logic. Consider a subset F of U. The functions from F can be combined by attaching values (outputs) of certains functions to variables (inputs) of certain functions in an arbitrary way so that we obtain a single value and no feedback is created. This single value defines a unique function f of external variables to which no value of F functions is attached. We say that f is a composition (in the ordinary sense) of F functions. The closure of F is the set of all compositions of F functions and is denoted by [F]. The set F is complete in U if [F] = U, or equivalently if each function from U is a composition of F functions. A complete set F is a base

of U if it does not contain any complete proper subset. The unique function of a complete singleton set F is called a Sheffer function for U. More formal definition and description of these concepts will be given in section I.4.

The closure [C] of the set C = {$\cup$, $\cap$, $^-$, constants}, also denoted by BF, is referred to as the set of Boolean set logic functions *with* constants. The closure [B] of the set B = {$\cup$, $\cap$, $^-$} is referred to as the set of Boolean set logic functions *without* constants (the constants are not involved in the compositions of B functions). Consider any subset S of C and a subset F of non Boolean functions from U. It is well known that S is not complete, that is [S] $\neq$ U. However, Boolean functions are convenient choice as building blocks in the design of carrier computing circuits; they are considered to be very cheap elements, i.e. they are usually obtained with small cost considering the complexity of their implementation versus that of non Boolean functions. A variation of the definition of completeness is then the concept of *complete with S functions*, abbreviated *S-complete*, which assumes that for composition besides a F function one can freely utilize S functions. More precisely, a subset F of U is S-complete in U if F $\cup$ S is complete in U. In some literature, the notions of *Boolean completeness* or *weak completeness* are used, with the same meaning as S-completeness (for [S] containing Boolean functions). The concept of S-completeness brings new interesting questions on the properties of S itself. For instance, given two distinct subsets of C, they may lead exactly or approximately to the same S-completeness criteria, and the subset with the smallest cardinality is probably the most interesting for engineers who develop efficient carrier computing chips or circuits. There are many others questions.

The notions of C-completeness and B-completeness of a set F of non Boolean set logic functions were introduced. The C-completeness (B-completeness) of F is defined as the completeness once all C functions (B functions, resp.) are added to F. Previous researches gave full description of C-maximal sets and B-maximal sets, C-bases and C-Sheffer functions for the case two-valued and three-valued set logic. Our thesis tends to be a continuation of these previous researches. We were also motivated by the possibility to apply computer in completeness problem, which is a new approach while previous results in this domain were obtained mostly by mathematical means.

Contributions

Our research focuses on the classification and enumeration of functions and bases of two-valued and three-valued set logic under compositions with B functions.

1. We collect from distinct literatures important studies relevant to the design of carrier computing systems. We obtain a survey of all known results in set logic. We first mention results in completeness theory (functional completeness, S-completeness and Boolean completeness). Second, we describe a class of primal algebras called bio-algebras that model biological molecular computing. Finally, we discuss results in approximation theory.

2. We develop algorithms for classification of (set logic) functions, for enumeration of bases (in set logic) and for generation of Boolean (set logic) functions. These algorithms are very useful for the study of completeness properties (of set logic).

3. Using our algorithms, we give computational results on classification of one-place functions and enumeration of bases (containing one-place functions) of two-valued and three-valued set logic under compositions with C functions. The results on classification of functions agree with theoretical results mentioned in [D1,S3]. The results on enumeration of (classes of) C-bases are new.

4. We discuss some classification and enumeration problems in r-valued set logic. B-maximal sets are maximal sets containing all B functions. We give the number of n-place functions in each B-maximal set and find some properties of intersections of B-maximal sets in r-valued set logic. These properties are used to classify all r-valued set logic functions according to the B-maximal sets they belong to, and also to enumerate the B-Sheffer functions and the (classes of) B-bases in r-valued set logic.

5. Our main contributions are the followings. We prove, computationally and theoretically, that there are 8 and 200 classes of functions respectively in two-valued and three-valued set logic under B functions compositions. For each class of functions we give a one-place example function and its total number of one-place set logic functions.

6. We find an upper bound on the number of n-place [B] functions (i.e. Boolean functions without constants) in r-valued set logic.

7. We study the B-Sheffer functions, i.e. non Boolean set logic functions from singleton sets that are complete and irredundant under compositions with B functions. We find the number of n-place B-Sheffer functions of two-valued set logic and give a lower bound and an upper bound on the number of n-place B-Sheffer functions of three-valued set logic.

8. We enumerate all the (classes of) B-bases containing one-place functions of two valued and three-valued set logic, and find that the maximum rank of a B-base is 3 and 7 for r =2 and r = 3, respectively.

The contributions 1, 2 and 3 are made solely by the thesis author. All others contributions are obtained jointly with Pr. C. Reischer and Pr. I. Stojmenovic and are reported in a paper submitted for publication [N1].

CHAPTER I
SET LOGIC ALGEBRA

I.1. Concept of set logic

The works on carrier computing suggested the interest to study set logic functions and the complexity of their implementation using Boolean and non Boolean switching devices. Set logic algebra (SLA) is a special class of multiple-valued logic algebra. It was proposed first as a new foundation of the biological molecular computing in [A7]. More formal algebraic aspects of SLA has been studied in [B2,D1,D2,N1,R1,R2,S3,S4,T1].

We consider the set $R = \{e_0, e_1, ..., e_{r-1}\}$ as the set of fundamental values. The carrier computing circuits operate on the power set of R, i.e. the set of all subsets of R we denote by $2^R = \{X \mid X \subseteq R\}$. An element $X \in 2^R$ is called a logic set. Thus, if R contains r logic values then 2^R contains 2^r logic sets. The basic operations over 2^R are the set-theoretic union ($\cup$), intersection ($\cap$) and complement ($\bar{}$) operations. It is well known that 2^R is a Boolean algebra $[2^R, \varnothing, R, \cup, \cap, \bar{}]$ when equipped with these three set-theoretic operations.

Mathematically, the SLA is based on the isomorphism relationship between the Boolean algebras $[B^r, 0, 1, +, \cdot, ']$ and $[2^R, \varnothing, R, \cup, \cap, \bar{}]$. B^r is the Cartesian product of r 2-elements Boolean algebras $B = \{0, 1\}$. A carrier computing circuit can be described as a set logic function of n variables, which is a mapping

$$f: (2^R)^n \rightarrow 2^R$$

that map n-tuples of subsets of R into a subset of R. The set of all such functions is referred to as r-valued set logic. The number of n-place set logic functions is quite considerable: there are $(2^r)^{(2^r)^n}$ such functions [A7]. As the radix r increases, this number becomes enormously large. For example, for $n = 1$ and $r = 2$ there are 256 functions while for $n = 1$ and $r = 3$ we find 16777216 such one-place functions.

A small fraction of these functions are Boolean functions, that is functions that can be constructed from constants and variables, using union, intersection and complementation. The number of n-place r-valued Boolean functions of set logic is $(2^r)^{2^n}$ (see section II.4). For instance, for $r = 2$ and $r = 3$ we find 16 and 64 (resp.) one-place Boolean set logic functions.

I.2. Set logic algebra as a class of multiple-valued logic algebra

Another way of looking to set logic functions is considering them as functions over a logic with 2^r values which provides a rich collection of functions over logic with very high radix. We shall see that there are certain computational advantages in adopting this dual point of view.

Consider the following situation arising in the synthesis of switching circuits. We have certain basic elements called gates. Each gate has one or several *inputs* and a single *output*. The gate receives signals on the inputs and transforms them into the output signal. For simplicity we assume that all signals belong to the same finite set denoted by $L_k = \{0, 1, ..., k - 1\}$, $k \geq 1$. The functioning of the gate can be described by the assignment of the output value f(x) to every ordered n-tuple $x = (x_1, ..., x_n)$ of input values. Thus the gate realizes a function f of n variables ranging in the finite set L_k with values in L_k. In other words f maps the *cartesian power* L_k^n (of all ordered n-tuples of elements of L_k) into L_k. Denote by $P_k(n)$ the set of all such functions. Thus $P_k(n)$ consists of $|P_k(n)| = k^{k^n}$ functions. The set $P_k(n)$ is called the set of *n-ary operations* on L_k in universal algebras and the set of *n-ary functions of k-valued logic* in multiple-valued logic algebras. The set of n-place k-valued logic functions $f: L_k^n \rightarrow L_k$ is denoted by $P_k(n)$. The union of $P_k(n)$ for $n = 1, 2, ...,$ is denoted by P_k.

As discussed in [S3], every r-valued set logic function can be regarded as a k-valued logic function for $k = 2^r$, as follows. Without loss of generality we may use characteristic binary vectors to represent the elements of 2^R as binary numbers. A subset $X \in 2^R$ is represented as binary number $x_0x_1...x_{r-2}x_{r-1}$ determined by $x_i = 1$ if and only if $e_i \in X$, for $i = 0, 1, ..., r - 1$. Next, X is mapped into the natural number x which has binary representation $x_{r-1}x_{r-2}...x_1x_0$, i.e. $x = 2^{r-1}x_{r-1} + 2^{r-2}x_{r-2} + ... + 2x_1 + x_0$. We also refer to x_i, $i = r - 1, ..., 0$ as the i-th coordinate of x.

For instance, for $r = 2$ we have $R = \{e_0, e_1\}$ and the elements of $2^{\{e_0, e_1\}}$ are represented in the following way:

Set X	Binary x_1x_0	Decimal x
$\varnothing$	00	0
$\{e_0\}$	01	1
$\{e_1\}$	10	2
$\{e_1, e_0\}$	11	3

The operations $\cup$, $\cap$ and $\bar{}$ are represented by the following tables.

$\cup$	0	1	2	3
0	0	1	2	3
1	1	1	3	3
2	2	3	2	3
3	3	3	3	3

$\cap$	0	1	2	3
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

$\bar{}$	
0	3
1	2
2	1
3	0

In general, $x \cup y = u$ and $x \cap y = v$ are determined by $u_i = \max(x_i, y_i)$ and $v_i = \min(x_i, y_i)$ for $i = 0, 1, ..., r - 1$, while $\bar{x} = k - 1 - x$. We refer to these functions as union, intersection and complement functions in P_k, $k = 2^r$. For example, for $r = 3$ (i.e. $k = 8$) we have $3 \cup 5 = 011 \cup 101 = (0 \cup 1)(1 \cup 0)(1 \cup 1) = 111 = 7$, $3 \cap 5 = 011 \cap 101 = (0 \cap 1)(1 \cap 0)(1 \cap 1) = 001 = 1$ and $\bar{3} = \overline{011} = 100 = 4 = 8 - 1 - 3$.

I.3. Boolean set logic functions

The set of Boolean set logic functions is denoted by BF or [C] and contains all functions obtained from the set $C = \{\cup, \cap, \bar{}, c_0, c_1, ..., c_{k-1}\}$ by composition (where c_i is the constant function i). The set of Boolean functions constructed from the set $B = \{\cup, \cap, \bar{}\}$ is denoted by [B] (constants are not involved in their composition). Also, let $BF_k(n)$ be the set of all Boolean n-place functions in P_k. The number of such functions is $|BF_k(n)| = k^{2^n}$, where $k = 2^r$. For example $|BF_4(1)| = 16$, $|BF_8(2)| = 4096$. [C] ([B]) is also referred to as the set of Boolean functions **with** constants (**without** constants, resp.).

The symmetric difference over 2^R will be denoted by $\oplus$ and is defined as $X \oplus Y = (X \setminus Y) \cup (Y \setminus X)$. If $X, Y \in 2^R$, then $X^0 = R$, $X^1 = X$ and $XY = X \cap Y$. As it was observed in [R7], pp.37, a (set logic) function b is Boolean if and only if it can be written as

$$b(X_1, ..., X_n) = \sum_{(i_1, ..., i_n)} A^n_{i_1, ..., i_n} X_1^{i_1} ... X_n^{i_n}$$

for every $X_1, ..., X_n \in 2^R$, where $A^n_{i_1, ..., i_n}$ are constants of 2^R while the sum is extended over all binary numbers $i_1...i_n$ (i.e. $i_j \in \{0, 1\}$ for $1 \le j \le n$) between 0 and $2^n - 1$. In the above formula, the sum represents the extension of $\oplus$. For the case $n = 2$, for example, the indices range between 0 and 3 and we have

$$b(X_1, X_2) = A^2_{00}X^0_1X^0_2 \oplus A^2_{01}X^0_1X^1_2 \oplus A^2_{10}X^1_1X^0_2 \oplus A^2_{11}X^1_1X^1_2$$
$$= A^2_0 \oplus A^2_1X_2 \oplus A^2_2X_1 \oplus A^2_3X_1X_2.$$

Almost all set logic functions are non Boolean. That is they are not polynomials of the Boolean algebra of 2^R, since they cannot be written in the canonical polynomial form. Therefore, we cannot construct all the set logic functions from any subset of C by composition, i.e. subsets of Boolean set logic functions are not complete. On the other hand, Boolean set logic functions are very cheap functions. In switching circuits usually the Boolean functions are at our disposal at small cost, i.e. they can be realized using binary standard electronic circuitry.

If the elements of 2^R are represented as binary numbers, then if $x, y \in L_k$, $x \oplus y$ is determined by $w_i = 0$ if $x_i = y_i$ and $w_i = 1$ if $x_i \neq y_i$, for $i = 0, ..., r - 1$. We refer to this operation as the *exclusive or* operation in P_k, $k = 2^r$. We have the following table for $r = 2$.

⊕	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

If $x, y \in L_k$, then we write $xy = x \cap y$ (operation in P_k). Therefore, a (set logic) function b is Boolean if and only if it can be written as

$$b(x_1, ..., x_n) = a_0 \oplus \sum^{1, ..., n}_{i_1...i_m} a_{i_1...i_m} x_{i_1} ... x_{i_m} \quad [R7]$$

where a_0 and $a_{i_1...i_m}$ are constants of L_k while the sum is extended over all subsets $\{i_1, ..., i_m\}$ of m distinct indices from the set $\{1, ..., n\}$. The sum represents the extension of the operation $\oplus$ over L_k. The coefficients a_0 and $a_{i_1...i_m}$ are uniquely determined by the function b. For instance, $b(x_1, x_2) = a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus a_{12}x_1x_2$. This canonical polynomial form is sometime called the Rudeanu formula or the Zhegalkin polynomial.

I.4. Completeness and approximation properties in set logic

Completeness and approximation properties in set logic are of great importance to designers of carrier computing systems because they allow the reduction of the implementation complexity of the carrier computing circuits.

I.4.a. Completeness properties

Investigations of completeness and related topics, which are usually called functional completeness problems are directly related to logic circuit design, and they have a wide area of applications in addition to their mathematical importance.

The compositions of functions from a set F of (set logic) functions are precisely the functions that can be realized by combinatorial switching or logic circuits constructed from gates with transformation functions from F. A gate is of *type* f (or realizes f) if it transforms any input (x_1, x_2, ..., x_n), $x_i \in L_k$, into a single output $f(x_1, x_2, ..., x_n)$ depending on x_1, x_2, ..., x_n only. Thus the type of a gate is completely described by a single function of n variables ranging over L_k and with all values in L_k. Consider a collection of gates of type $f_i \in P_k$. These gates can be combined into logic circuits by attaching outputs of certain gates to inputs of certain gates in an arbitrary way so that the resulting circuit has a single output and no feedback is created. This means that the single output of such a circuit defines a unique function $f \in P_k$ of external inputs to which no gate is connected. This function can be described as the composition (or superposition) of the f_i's.

The compositions includes permuting variables in a function, identifying two variables, and replacing variables by functions from F. If the functions from F are treated as circuits then the composition is the creation of new circuits by using the output of some circuits as input to other ones, where it is allowed to use multiple copies of same output or to permute input *wires*. The closure of F is denoted by [F] and contains all compositions of functions from F. It is natural to ask the following question: What are the properties of [F] ?

A subset F of P_k is said to be closed if it contains all compositions (or superpositions) of its members [J1,R3], i.e. if the formation of composition does not lead outside F, or equivalently, if [F] = F. More precisely, for $f \in P_k$ m-ary and $g \in P_k$ n-ary put $s := m + n - 1$ and define the s-ary $h = f * g$ by setting (for instance)

$h(x_1, x_2, ..., x_s) := f(g(x_1, x_2, ..., x_n), x_{n+1}, ..., x_s)$ (substitute $(x_1, ..., x_n)$ by $g(x_1, ..., x_n)$)

for all $x_1, x_2, ..., x_s \in L_k$. Further, for $m > 1$ put (for instance)

$$(\zeta f)(x_1, x_2, ..., x_m) := f(x_2, x_3, ..., x_m, x_1), \quad \text{(shift operation)},$$
$$(\tau f)(x_1, x_2, ..., x_m) := f(x_2, x_1, x_3, ..., x_m), \quad \text{(permute } x_1 \text{ and } x_2\text{)},$$
$$(\Delta f)(x_1, x_2, x_3, ..., x_m) := f(x_1, x_1, x_3, ..., x_m) \quad \text{(replace } x_2 \text{ by } x_1\text{)}$$

for all $x_1, ..., x_m \in L_k$ and $\zeta f = \tau f = \Delta f = f$ for f unary ($m = 1$). Now F is closed if $f * g$, ζf, τf and $\Delta f \in F$ whenever $f, g \in F$. It is well known that the set of all closed sets ordered by inclusion is an algebraic lattice.

For closed sets F such that $F \subset P_k$ (proper inclusion), F is a P_k-maximal set if there is no closed set G such that $F \subset G \subset P_k$ (i.e. P_k covers F in the set of closed sets ordered by $\subseteq$). That is F cannot be properly extended to a closed subset G of P_k.

The first and most natural problem is the characterization of subsets F of P_k such that $[F] = P_k$. Such sets are called (functionally) complete in P_k. The algebras $\langle L_k, F\rangle$ with F complete in P_k are called primal algebras. Thus F is complete if and only if any function of P_k can be realized by a circuit constructed exclusively from gates realizing functions from F. That is, a subset F of P_k is complete in P_k if P_k is the least closed set containing F (in other words, if the functions in F can produce by composition any function in P_k).

A complete set F in P_k is called a base of P_k if no proper subset of F is complete in P_k. The rank of a base is the number of its elements. A function f is Sheffer for P_k if $\{f\}$ is a base (of rank 1) of P_k.

Unfortunately, it is well known that the Boolean algebra on 2^R, say $[2^R, \varnothing, R, \cup, \cap, \bar{}\,]$ is not functionally complete except for the case $r = 1$. That is, for $r \geq 2$ we cannot produce all the $(2^r)^{(2^r)^n}$ set logic functions using only the $\cup$, $\cap$, $\bar{}$ and the constants functions. In order to achieve the functional completeness in set logic, non Boolean set logic functions are added to some set S of Boolean set logic functions. Example of complete sets with constants over 2^R are discussed in literature such as the sets $\{\cup, \bar{}, \text{Conversion}\}$ [M1], $\{\cup, \setminus \text{ or Bio-Pass, Generation or Bio-Output}, \bar{}\}$ [A3], and $\{\cup, \cap, \text{Literal}\}$ [Y2]. All the $(2^r)^{(2^r)^n}$ set logic functions can then be expressed using only the operators of one of these complete sets. The Conversion, Bio-Pass, Bio-Output and Literal functions are given respectively by

$$X^{\sigma} = \bigcup_{\varepsilon_i \in X} \sigma(\varepsilon_i),\ \sigma: R \to 2^R,$$

$$BP(X, Y) = X \setminus Y,$$

$$BO(X, Y) = \begin{cases} X & \text{if } Y = \varnothing \\ \varnothing & \text{otherwise} \end{cases},$$

$$A_X B = \begin{cases} R & \text{if } A \subseteq X \subseteq B \\ \varnothing & \text{otherwise} \end{cases},$$

for all A, B, X, Y $\in 2^R$. We will discuss details on these functions in sections III.3, III.4 and IV.5.

I.4.b. Approximation properties

The approximation properties offer the possibility of building hybrid circuits, that is circuits that use binary electronic components and non-binary carrier computing components which correspond to Boolean and non Boolean functions, respectively.

If f is a one-place Boolean (set logic) function then, for any X, Y $\in 2^R$ we have

$$f(X) \oplus f(Y) \subseteq X \oplus Y.$$

Indeed, there exist A, B $\in 2^R$ such that $f(X) = A \oplus BX$, for every X $\in 2^R$. This, in turn implies $f(X) \oplus f(Y) = A \oplus BX \oplus A \oplus BY = B(X \oplus Y) \subseteq X \oplus Y$, for any X, Y $\in 2^R$. This property of Boolean function was obtained in [S1] and generalized in [R7].

A function F $\in P_k(1)$ generates an equivalence $\sim_F$ on 2^R. Namely, whenever ($F(X) \oplus F(W) \subseteq X \oplus W$ if and only if $F(Y) \oplus F(W) \subseteq Y \oplus W$) for every W $\in 2^R$, then X $\sim_F$ Y. The equivalence class of X with respect to $\sim_F$ will be denoted by $[X]_F$. The equivalence classes of the quotient set $2^R/\sim_F$ are exactly those sets on which we can *approximate* the set logic function F by Boolean functions [R1] (see section IV.6.a). Let F, G $\in P_k(1)$. F and G are equivalent if the one-place set logic function h given by $h(X) = F(X) \oplus G(X)$ is a Boolean function. We denote this by F ≡ G.

[T'1] introduced the notion of Boolean collection which appears to play a role in the approximation of non Boolean set logic functions by Boolean set logic functions. A non empty collection of sets $C \subseteq (2^R)^n$ is a Boolean collection if there exists a Boolean set logic function f such that $C = C_f$ where $C_f = \{X \in (2^R)^n \mid f(X) = \varnothing\}$.

Any singleton set $\{L\}$ is a Boolean collection, where $L \in 2^R$ since $\{L\} = C_f$, where $f(X) = X \oplus L$ for $X \in 2^R$. The set $(2^R)^n$ is a Boolean collection defined by the Boolean set logic function $f(X_1, ..., X_n) = \varnothing$ for $X_1, ..., X_n \in 2^R$. Note that for any set $H \in 2^R$ and any Boolean set logic function f the collection $C = \{X \in (2^R)^n \mid f(X) = H\}$ is Boolean since we can write $C = \{X \in (2^R)^n \mid g(X) = \varnothing\}$, where g is a set logic function given by $g(X) = f(X) \oplus H$ for all $X \in (2^R)^n$.

The set of Boolean collections on $(2^R)^n$ is closed with respect to intersection. Indeed, if f, g are two Boolean set logic functions, it is easy to verify that $C_f \cap C_g = C_{f \cup g}$, where $(f \cup g)(X) = f(X) \cup g(X)$ for every $X \in (2^R)^n$. This allows for the consideration of the least Boolean collections that contains an arbitrary collection of sets. Also, note that if f, g are two Boolean set logic functions, then their equalizing collection $C_{f=g} = \{X \in (2^R)^n \mid f(X) = g(X)\}$ is also a Boolean collection since $C_{f=g} = C_{f \oplus g}$, where $(f \oplus g)(X) = f(X) \oplus g(X)$ for every $X \in 2^R$.

CHAPTER II
CARRIER COMPUTING

II.1. Introduction

Interconnection problems are recognized to be a basic limitation in the present-day VLSI systems. VLSI circuit technology has seen rapid advancement resulting in dramatic improvements in both performance and cost per function of integrated circuit. Significant evolution in processing and design methodology has led to an increase in chip density and complexity, such that in complex VLSI systems, interconnections cause serious problems such as noises, added delay and power consumption. This is because the effective chip area devoted to device interconnects far exceeds that occupied by active devices; chip area and chip size are mostly dominated by wiring rather than active devices. Chip cost, performance and speed are determined by interconnect delay and area. Therefore, minimal and local interconnections become an essential factor for an effective realization of VLSI circuits [A2]. However, many computationally demanding problems for high speed VLSI applications (such as intelligent robot control or high speed image processing for instance) require global interconnections inherently to perform tasks in highly parallel way. It is reported in [A2] that in general, such highly parallel processing architectures with large-scale interconnection network, require almost $\Omega(n^2)$ wire area for n processing elements. Consequently, this results in serious interconnection bottlenecks.

A solution to the above-mentioned problems is to implement the concept of *multiplex computing* which reduces the number of interconnections by passing more information through the signal lines, i.e. increasing the information density. The term *multiplex computing* implies that many independent computational activities are stuffed into a single line, and performed in parallel based on multiplex data transmission. A typical example of multiplex computing in conventional VLSI architecture is the use of more than 2 levels of logic, namely, the *multiple-valued logic* (*MVL*) system [W1].

In the MVL system, the processing capability increases as the radix (or the base) increases. For example, in a 16-valued logic system, a single MVL module can operate 4 binary functions simultaneously, as if 4 binary logic modules were multiplexed into the 16-valued module. In general, a 2^r-valued module is needed to multiplex r functions in the MVL system: that is, a 2^r-valued function can simulate r binary functions simultaneously as if it were r binary functions operating in parallel. This is the fundamental principle of multiplex computing. However, for

highly parallel computing architectures, it is still difficult at the present state of technology to design MVL systems with very high radix such as 1024 (= 2^{10}), with reasonable stability and precision.

II.2. Multiplexing with information carriers

The term *multiplexable information carrier* is used to refer to any entity carrying information in its milieu, and can be multiplexed. An information carrier is also called an *elementary carrier*. Three types of information carriers are presented later in section II.5: the electric frequencies, the optical wavelengths, and the biological molecules.

Let the set of logic values for multiplex computation in ultra higher-valued logic system be given by $L = \{0, 1, ..., r - 1\}$. Let $E = \{c_0, c_1, ..., c_{r-1}\}$ be the set of r multiplexable information carriers. In the following discussion, we assume a one-to-one correspondence between the r logic values and the r kinds of elementary carriers as

$$\text{logic value } i \leftrightarrow \text{elementary carrier } c_i, \qquad i = 0, ..., r - 1.$$

Thus, the logic value $i \in L$ is represented by the presence of the elementary carrier $c_i \in E$, so that *logic value multiplexing* can be achieved. In all realizable logic systems, whether binary or non-binary, the most important property is that of logic value integrity, namely, generation, transmission and detection [A2] of logic values. From this view point, the following property for multiplexable information carriers must be assumed:

1. *Generation*: each elementary carrier c_i (logic value i) can be produced selectively within a multiplex computing network.

2. *Transmission*: the information carriers (logic values) can be transmitted, without interaction, through a media. Each elementary carrier c_i represents 1-bit information by its presence (bit 1) or absence (bit 0) in the media.

3. *Detection*: a specific elementary carrier c_i (logic value i) can be easily discriminated from the other elementary carriers.

To show how the information carriers can be multiplexed, let us consider the example of any *carrier-computing* (molecular, frequency, optical, ...) of four "AND" functions. Assuming that

the set of elementary carriers E = $\{c_0, c_1, c_2, c_3\}$, the simple multiplex computing module schematically illustrated in figure 1a, combines two multiplexed signals into one. When a specific elementary carrier c_i enters into both the two input ports, the same elementary carrier c_i exits from the circuit.

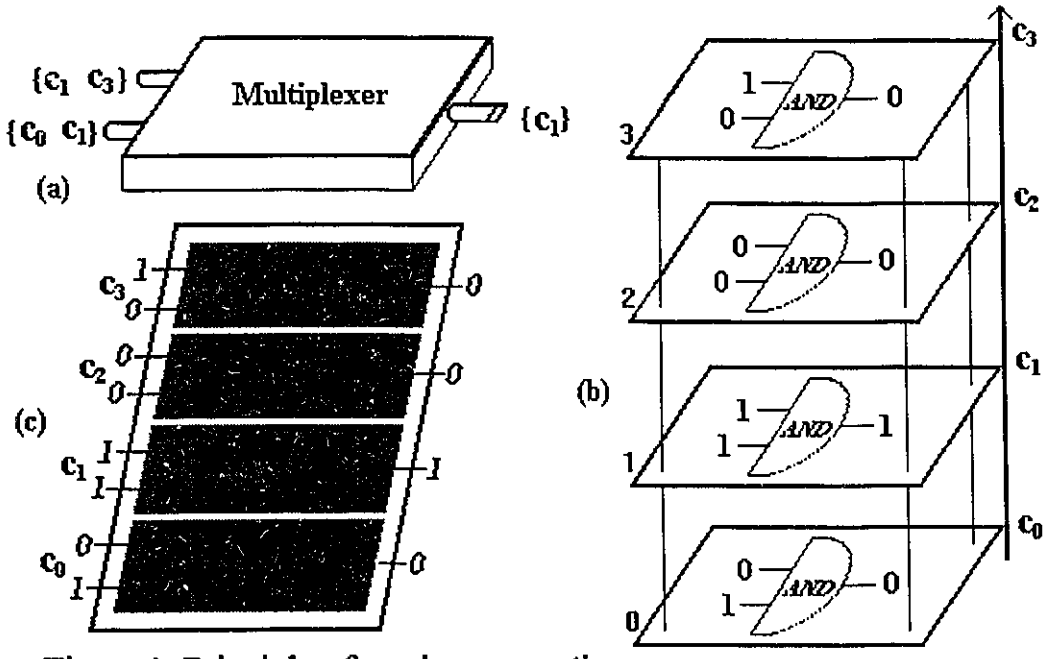


Figure 1: Principle of carrier computing:
(a)Multiplexer (b)Carrier-space diagram
(c)Concurrency with a conventionnal VLSI circuit

From another point of view, if we assign a distinct elementary carrier (logic value) to each AND gate of figure 1c, the presence of each elementary carrier in figure 1a represents the result of four independent binary operations as follows:

$$c_0 : 1 \cdot 0 = 0, \qquad c_1 : 1 \cdot 1 = 1, \qquad c_2 : 0 \cdot 0 = 0, \qquad c_3 : 0 \cdot 1 = 0.$$

where $\cdot$ represents the binary AND operation, and the binary logic values 1 and 0 correspond respectively to the presence and the absence of the elementary carrier c_i in the media. The binary logic value 1 means that the given elementary carrier assigned to the AND gate exists in the media, and the binary logic value 0 means that it doesn't exist. For example, c_3 is assigned to the fourth AND gate, and as the binary logic value 1 appears only in the second input, therefore c_3 must appear also only in the second input in figure 1a. If we write on the same row all 1st-input binary values together, all 2nd-input binary values together, and all output binary

values together, we obtain the following results (each column represents an AND gate assigned to an elementary carrier)

first inputs:	0101	→	c_1, c_3	or	1, 3	
second inputs:	1100	→	c_0, c_1	or	0, 1	(in terms of logic values)
outputs:	0100	→	c_1	or	1	

Thus the module of figure 1a performs four binary AND functions simultaneously as shown in figure 1b (or 1c). This figure is also called the *carrier-space diagram* in which the planes (the *carrier-planes*) visualizes the activity of the elementary carriers. One can see that the module of figure 1a performs simply the set-theoretic Intersection operation which is equivalent to the four binary AND operations of the circuit in figure 1c.

In the conventional VLSI module of figure 1c, four AND gates operating simultaneously are required to perform the four AND functions concurrently. Consequently, the principle of carrier computing allows us to reduce the number of interconnections to at most 1/r of the original conventional VLSI module (in our example, the number on interconnections is reduced from 12 to 3, for r = 4). However, if we multiplex a highly parallel architecture with global interconnections -the multiplexed binary modules are globally interconnected-, the reduction of the interconnection complexity become $1/r^2$ due to the reduction of total interconnections including interconnections between modules. Therefore we can say that, in general, the reduction of the interconnection area is between $1/r^2$ and 1/r inclusively. Due to this reduction, carrier computing offers a possibility of massively parallel computing without interconnection problems.

II.3. Algebraic foundation

In general, the term *multiplex computing* means that several binary computing modules are multiplexed into a single module that performs the corresponding binary logic functions in parallel using the parallelism of multiplexable information carriers. For the systematic design of carrier computing systems, a new algebraic system that is the *set-valued logic* (*SVL*) system, has been proposed, in which multiplexed binary signals are treated as sets of elementary carriers. For example, the parallel operation of our multiplexer in figure 1a can be represented simply by the set-theoretic intersection operation $\{c_0, c_1\} \cap \{c_1, c_3\}$, or in term of logic values $\{0, 1\} \cap \{1, 3\}$. This is a simple example of operation in set logic algebra, where the input and the

output values are set of logic values (i.e. logic sets). Therefore, since the elementary carriers are treated as set of logic values the concept of set logic is very suitable for carrier computing.

An SVL system is an ultra higher-valued logic system in which, instead of 2^r-valued logic function, only a r-valued function is used to multiplex r binary functions. We can perform very high degree of multiplexing (i.e. using high radix such as 1024 or 2^{20} !). Thus, one can appreciate the big difference between the SVL and the MVL systems. With the SVL system, a great reduction of interconnections (more than in the MVL system) can be achieved, so that the *ultra* highly parallel architectures can be designed. In general, in the SVL module, assuming that r elementary carriers c_0, c_1, ..., c_{r-1} are available, r logic operations are done in parallel, that is information contained in a single line becomes r bits. Considering the transmission of r logic values, table 1 shows the comparison of the SVL system, the MVL system and the conventional binary logic system according to information density [A7].

Table 1 : Logic systems	Information contained in a single line
Binary logic system	1 bit
r-valued logic system	$\log_2 r$ bits
r-valued set logic system	r bits

The concept of set logic enables us to handle the parallelism of multiplex computing intuitively. It is a switching algebra for computing based on multiplexable information carriers. Mathematically, the SVL system is based on the isomorphism relationship between the Boolean algebras $[B^r, 0, 1, +, \cdot, ']$ and $[2^L, \varnothing, L, \cup, \cap, \bar{}\,]$. B^r is the Cartesian product of r 2-elements Boolean algebras $B = \{0, 1\}$ (see section I.1). The power set 2^L denotes all the possible combinations of carriers transmitted through a line simultaneously.

II.4. Design principles of carrier computing system

In fact, the binary logic modules in a conventional VLSI system can be more complex than that of figure 1c. One typical binary module may contains a variation of any number of primitive binary gates such as OR, AND, and NOT gates. Distinct or identical gates or modules operate in parallel and communicate to each other. Thus, the binary modules are either independent or interdependent, and either homogeneous or heterogeneous. These binary modules can be multiplexed into one or more SVL modules, so that all these possibilities mentioned above can be achieved with the SVL system with much lower interconnection complexity. If the binary modules are homogeneous, a single common SVL module can be derived, otherwise if they are heterogeneous, several SVL modules may be derived. The number of SVL modules to derive

from the binary modules may also depends on the maximum degree of multiplexing, i.e. the number r of logic values which is the number of available elementary carriers. In any case, the choice of the primitive SVL gates is important for efficient design of ultra higher-valued logic architectures.

The choice of the primitive SVL gates that will serve to implement the set logic functions expressed by the complete set may depend on the complete set itself or also on the type of information carriers used to convey information, whether they are biological molecules, optical wavelengths, electric frequencies or others kinds. The type of information carriers determines the facility (or ability) in the carrier computing system to design the primitive SVL gates which can be Boolean or non Boolean. The choice of the complete set system itself may also depends on the type of multiplexing which is either homogeneous (the multiplexed binary modules are identical) or heterogeneous (the multiplexed binary modules are different) [A2]. One must keep in mind that the complete sets are interchangeable, any complete system can be used for any carrier computing system; the choice of one complete system in place of another one is only a matter of ability or facility.

Conventional VLSIs are composed of binary logic modules. These modules use binary variables whose the two possible values are represented by the logic values 0 and 1, and perform binary logic functions generally defined as a mapping

$$f : B^n \rightarrow B$$

On the other hand, carrier computing systems are composed of set logic modules. These modules use r kinds of elementary carriers to convey information, and perform set logic functions generally defined as a mapping

$$F : (2^L)^n \rightarrow 2^L$$

Let us consider r functional binary modules m_0, m_1, ..., m_{r-1}. Each module m_i performs the binary logic function $y_i = f_i(x_{i1}, x_{i2}, ..., x_{in})$, i.e.

$$\begin{array}{llllllll}
m_0: & y_0 & = & f_0(x_{01}, & x_{02}, & ..., & x_{0n}) \\
m_1: & y_1 & = & f_1(x_{11}, & x_{12}, & ..., & x_{1n}) \\
. & . & . & . \quad . & . & . & . \\
m_{r-1}: & y_{r-1} & = & f_{r-1}(x_{r-11}, & x_{r-12}, & ..., & x_{r-1n})
\end{array}$$

Mathematically, the design of carrier-computing circuits can be formulated as the multiplexing of a set of binary functions $f_0, f_1, ..., f_{r-1}$ into a functionally equivalent SVL function F, using the *isomorphic transformation*

$$\varphi : B^r \rightarrow 2^L$$

$$\text{as } \varphi(x_{0k}, x_{1k}, ..., x_{r-1k}) = X_k = \{i \mid x_{ik} = 1, i \in L\}, \qquad k = 1, ..., n$$

Applying this transformation φ we obtain the specification of the SVL functional module M into which $m_0, m_1, ..., m_{r-1}$ are multiplexed as

$$M : Y = F(X_1, X_2, ..., X_n), \qquad \text{where } Y = \varphi(y_0, y_1, ..., y_{r-1}).$$

For example, each gate in figure 1c performs the function $y_i = and_i(x_{i1}, x_{i2}) = x_{i1} \cdot x_{i2}$ as

$$\begin{aligned} m_0 &: \quad y_0 = 0 = and_0(1, 0) \\ m_1 &: \quad y_1 = 1 = and_1(1, 1) \\ m_2 &: \quad y_2 = 0 = and_2(0, 0) \\ m_3 &: \quad y_3 = 0 = and_3(0, 1) \end{aligned}$$

Using the isomorphic transformation we obtain $Y = \varphi(0, 1, 0, 0) = \{1\}$
$X_1 = \varphi(1, 1, 0, 0) = \{0, 1\}$
$X_2 = \varphi(0, 1, 0, 1) = \{1, 3\}$
Thus the SVL module M: $Y = \{1\} = F(X_1, X_2) = AND(\{0, 1\}, \{1, 3\}) = \{0, 1\} \cap \{1, 3\}$.

Each binary module m_i has n binary input variables $x_{i1}, ..., x_{in}$ and one binary output variable y_i. Each variable has the value 0 or 1 and therefore the number of binary logic functions the module m_i realizes is 2^{2^n}. Since r binary modules (functions) are multiplexed into a single SVL module M (set logic function) then the number of SVL functions F we make is $(2^{2^n})^r = |BF_k(n)|$ which is the number of Boolean n-place set logic functions in P_k, $k = 2^r$. On the other hand, binary logic functions are Boolean functions and then can be expressed using the Rudeanu formula. Using the canonical polynomial form of a Boolean function and the isomorphic transformation we prove that all the $(2^{2^n})^r$ set logic functions are Boolean. Consider for simplicity the case r = 2 and n = 1 (the proof is the same for all cases). Each binary module will realize the binary function $y_i = f_i(x_{i1}) = a_{i0} \oplus a_{i1}x_{i1}$, i.e.

$$m_0: \quad y_0 = f_0(x_{01}) = a_{00} \oplus a_{01}x_{01}$$
$$m_1: \quad y_1 = f_1(x_{11}) = a_{10} \oplus a_{11}x_{11}$$

where the a_{ij} are constant over {0, 1} and the $\oplus$ operation is the *exclusive or* operation on {0, 1}. Since B^r and 2^L are isomorphic then we obtain $\varphi(a_{00} \oplus a_{01}x_{01}, a_{10} \oplus a_{11}x_{11}) = \varphi(a_{00}, a_{10}) \otimes \varphi(a_{01}x_{01}, a_{11}x_{11}) = \varphi(a_{00}, a_{10}) \otimes \varphi(a_{01}, a_{11}) \cap \varphi(x_{01}, x_{11}) = A_0 \otimes A_1 \cap X_1$. This is the canonical polynomial representation of a one-place Boolean set logic function (see section I.3), where the A_i are constant over 2^L, the X_i are variables over 2^L and the $\otimes$ operation corresponds to the symmetric difference operation on 2^L. Therefore, the one-place set logic function we obtain is always Boolean. Now we prove the general case: r n-inputs binary modules. Each binary module m_j performs the function $(0 \le j \le r - 1)$

$$y_j = f_j(x_{j1}, x_{j2}, \ldots, x_{jn}) = a_{j0} \oplus {}_j\sum\nolimits_{i_1 \ldots i_m}^{1, \ldots, n} a_{i_1 \ldots i_m} x_{i_1} \ldots x_{i_m}.$$

Applying the isomorphic transformation we obtain

$$\varphi(a_{00} \oplus {}_0\sum\nolimits_{i_1 \ldots i_m}^{1, \ldots, n} a_{i_1 \ldots i_m} x_{i_1} \ldots x_{i_m}, \ldots, a_{r-10} \oplus {}_{r-1}\sum\nolimits_{i_1 \ldots i_m}^{1, \ldots, n} a_{i_1 \ldots i_m} x_{i_1} \ldots x_{i_m}) =$$

$$\varphi(a_{00}, \ldots, a_{r-10}) \otimes \varphi({}_0\sum\nolimits_{i_1 \ldots i_m}^{1, \ldots, n} a_{i_1 \ldots i_m} x_{i_1} \ldots x_{i_m}, \ldots, {}_{r-1}\sum\nolimits_{i_1 \ldots i_m}^{1, \ldots, n} a_{i_1 \ldots i_m} x_{i_1} \ldots x_{i_m}).$$

Each ${}_j\sum\nolimits_{i_1 \ldots i_m}^{1, \ldots, n} a_{i_1 \ldots i_m} x_{i_1} \ldots x_{i_m}$ contains $2^n - 1$ terms (ex. $y \oplus z$ has two terms: y and z). Let d be the decimal equivalent of the binary number $i_1 \ldots i_m$, then ${}_j a_{i_1 \ldots i_m} x_{i_1} \ldots x_{i_m}$ is the d-th term of the sum (i.e. xor) for the function f_j, $1 \le d \le 2^n$. All the d-th terms taken one per function f_j are multiplexed as

$$\varphi({}_0a_{i_1 \ldots i_m} x_{i_1} \ldots x_{i_m}, \ldots, {}_{r-1}a_{i_1 \ldots i_m} x_{i_1} \ldots x_{i_m}) =$$
$$\varphi({}_0a_{i_1 \ldots i_m}, \ldots, {}_{r-1}a_{i_1 \ldots i_m}) \cap \varphi({}_0x_{i_1} \ldots x_{i_m}, \ldots, {}_{r-1}x_{i_1} \ldots x_{i_m}) =$$
$$\varphi({}_0a_{i_1 \ldots i_m}, \ldots, {}_{r-1}a_{i_1 \ldots i_m}) \cap \varphi({}_0x_{i_1}, \ldots, {}_{r-1}x_{i_1}) \cap \ldots \cap \varphi({}_0x_{i_m}, \ldots, {}_{r-1}x_{i_m}) =$$
$$A_{i_1 \ldots i_m} \cap X_{i_1} \cap \ldots \cap X_{i_m}$$

As the *xor* operation is transformed into a set difference operation, then taking the sum (i.e. set diff.) of all terms $A_{i_1 \ldots i_m} \cap X_{i_1} \cap \ldots \cap X_{i_m}$ we finally obtain

$$\varphi(a_{00} \oplus_0 \sum_{i_1 \ldots i_m}^{1, \ldots, n} a_{i_1 \ldots i_m} x_{i_1} \ldots x_{i_m}, \ldots, a_{r-10} \oplus_{r-1} \sum_{i_1 \ldots i_m}^{1, \ldots, n} a_{i_1 \ldots i_m} x_{i_1} \ldots x_{i_m}) =$$

$$A_0 \otimes \sum_{i_1 \ldots i_m}^{1, \ldots, n} A_{i_1 \ldots i_m} \cap X_{i_1} \cap \ldots \cap X_{i_m}$$

which is the canonical polynomial representation of a n-place Boolean set logic function. The n-place set logic function we make are all Boolean and the set of $(2^{2^n})^r$ such functions is $BF_k(n)$. This does not mean that carrier computing refers only to Boolean set logic functions. Even if one cannot construct non Boolean functions from binary logic functions, a designer can still add (after multiplexing) non Boolean functions to its carrier computing circuits to achieve completeness (the literal function, for example, is added to the set $\{\cup, \cap\}$) and to have access to a rich set of set logic functions.

Thus, using the isomorphic transformation, the input binary variables and the output binary variables are respectively multiplexed into equivalent SVL input and output variables. The SVL function F thus obtained is then expressed using the operators of the proposed complete set, and finally implemented or designed using the proposed primitive SVL gates.

Assuming that r, i.e. the maximum degree of multiplexing, is greater than the number of modules in the binary network (so that all the binary modules can be multiplexed into a common SVL module), [Y1] gives the following general procedure for multiplexing r binary modules m_0, m_1, ..., m_{r-1} into a single common SVL module M, using r elementary carriers:

Step 1 : Assign distinct elementary carriers or logic values 0, 1, ..., r - 1 to each binary module $m_0, m_1, \ldots, m_{r-1}$.

Step 2 : Multiplex the binary variables in these modules into SVL variables in M using the isomorphic transformation φ, where the state of m_i is represented by the presence or absence of the corresponding elementary carrier c_i in M.

Step 3 : Replace primitive binary gates (OR, AND, NOT, XOR, NAND, NOR) with their SVL equivalents respectively, i.e. Union, Intersection, Complement gates, ...etc. Some

functions may usually be expressed using the operators of the complete set system. For instance, if the complete set system is {Union, Complement, Conversion}, occurrences of $\cap$ (Intersection) can be replaced by $\cup$ and $^{-}$ owing to the De Morgan's law: $X \cap Y = \overline{\overline{X} \cup \overline{Y}}$. Thus the SVL network can be realized using only the proposed primitive SVL gates.

Step 4 : Multiplex the interconnections between the binary modules. An interconnection between modules m_i and m_k corresponds to the interaction between elementary carriers c_i and c_k in M. Hence converting the carriers c_i to c_k realizes the connection. This cost-free data transmission is achieved using specific SVL operators such as the Conversion operator.

When the binary modules are heterogeneous, the multiplexing scheme may not be unique. The hidden regularity of these modules should be exploited. That is we can multiplex first each group of identical binary modules into temporary SVL modules, and after that, multiplex (if possible) these temporary SVL modules into one single SVL module as a result. Also, when the number of binary modules to be multiplexed is greater than the maximum degree of multiplexing, several modules should be derived. The way to determine carrier-planes (i.e. to assign the elementary carriers to the binary modules) and their interconnections should be considered. [M1] describes an algorithm based on vector projections. The problem to deal with, in general, is to find an optimal assignment of distinct elementary carriers to given binary modules; the reduction of the interconnections must be achieved without increasing the overhead inside the SVL module. That is, for the great reduction of the interconnection complexity, it is important to determine the optimal multiplexing scheme which minimizes the number of SVL operators and variables of a set logic function F.

II.5. Examples of carriers computing systems

II.5.a. Electric frequency multiplexing

The fundamental concept of logic values multiplexing can be achieved by using frequency of electric signals as the information carrier [Y1,Y2]. The logical information is coded into the combination of r kinds of frequency components $f_0, f_1, ..., f_{r-1}$ which are the elementary carriers. The logic value i is represented by the presence of the frequency component f_i. The r kinds of frequencies correspond to 2^r-valued representation, so that the ultra higher-valued logic

operations can be achieved. For instance, only 10 kinds of components (L = {0, 1, ..., 9}) are sufficient to represent 1024-valued logic information.

In this frequency computing, all the logical information is represented by the varieties of frequencies. Since these frequencies carry information by their presence or absence, the concept of frequency multiplexing can be essentially expressed in terms of set theory. From this point of view, the set-valued switching algebra is useful for the systematic design of frequency computing systems.

Something interesting here is the fact that [Y2] uses 2 complete set systems to design the SVL networks. First it uses the complete set S_1 = {Union, Intersection, Literal} for a conceptual realization of any SVL function. Second, the equivalent functional SVL module of this set logic function is realized using the primitive SVL gates from the complete set S_2 = {Union, Difference, Generation, Complement}. Because, it is easier to build the primitive Difference and Generation gates than to build the primitive Literal gate, using the available electronic devices such as filters, oscillators, and others. The Difference gate plays the role of a detector or discriminant function which selects a specific frequency and also the role of a logic value converter to multiplex the interconnections between binary modules. The Generation gate produces a specific set or frequencies. The set logic circuits are then more easily designed with primitive gates from S_2 than from S_1. In contrast, a SVL function can be easily expressed using the operators from S_1. This shows us that the complete systems are interchangeable.

II.5.b. Optical wavelength multiplexing

The basic concept of logic value multiplexing can also be achieved by optical wavelength multiplexing [M1]. The logic values are represented by the presence of the optical signals (the elementary carriers), each at different wavelength, so that several logic values can be simultaneously transmitted through a wavegide. Since a collection of optical signals is considered to be a set of logic values, the concept of set logic is very suitable for formulating the multiplex computation scheme. The complete system {Union, Complement, Conversion} is used for the systematic design of the optical computing system. The Conversion gate realizes the multiplexing of the interconnections between binary modules.

II.5.c. Biological molecules multiplexing

In [A3,A4,A5,A6,A7] there is a model of biological molecular computing where the switching devices (bio-devices) select and distribute specific biological molecules to perform highly parallel interconnection-free computing. The model is based on the high specificity of the enzymes -the biological catalysts- in their choice of the reactants called substrates. An enzyme recognizes specific substrate molecules in the fashion of key fitting into a lock. The substrates are broadcast in solution from sources. They diffuse with random molecular motion and carry information by their presence or absence in the solution. At specified destination, enzyme-based biosensors selectively detect the released substrates, which automatically triggers a specific biological molecular switch in the solution. In this sense, the data transmission and computation are interconnection-free.

In molecular computing, all the logical information is coded into varieties of biological molecules where each kind of substrate molecule (an elementary carrier) corresponds to a logic values. Since these biochemical substances carry information by their presence and absence, the concept of set-valued logic is then very useful to describe the operations. [A5,A7] use the complete set system {Union, Intersection, Literal} to express the set logic function, where the Literal function serves as a discriminant. However, they use distinct complete set systems to design the SVL modules and networks. In [A7] the complete set {Union, Bio-Pass, Bio-Output, Bio-Complement} is used, while [A5] uses the complete set {Union, BIO}. The Bio-Pass gate performs the logic values discrimination or selection function based on the enzyme-substrate reactions, while the Bio-Output gate is a generator that produces a specified set of logic values. The BIO gate performs the parallel selection and distribution functions. We discuss more details on these sets and on molecular computing in chapter III.

With the biological molecular computing system, *very* massive parallelism can be achieved, in comparison to frequency and optical multiplexing, because a great variety of elementary carriers are available. There are more than 10 thousand enzymes and 100 million antibodies (substrate molecules) in living systems. This new way of computing is completely interconnection-free -that is, computation is performed based on parallel distribution and parallel selection of biological molecules rather than on electronic switching.

To conclude this section, let's say that the electronic VLSIs systems are very effective for executing fast iterative or recursive arithmetic computation. Carrier computing systems have low data rates and slow switching speed, therefore they are not meant to compete with

electronic VLSIs systems in this area; this is their main disadvantage. Instead, their real advantage is massive parallelism. Also, this technology is not yet mature, several practical or technical difficulties remain still unsolved.

II.6. Some impacts of set logic

In this section we shall discuss some impacts of set logic in parallel sorting, parallel image processing, and other possible applications.

II.6.a. SVL-based parallel sorting network

A prominent example of application given in [M1] (optical wavelength multiplexing) and [A3] (biological molecule multiplexing), is the bitonic sorting network, which is of importance in the high-speed packet switching. The following definition and theorem provide the background necessary to understand this algorithm [A1]. A sequence $\{a_1, a_2, ..., a_{2n}\}$ is said to be bitonic if either (i) there is an integer $1 \le k \le 2n$ such that $a_1 \le a_2 \le ... \le a_k \ge a_{k+1} \ge ... \ge a_{2n}$ or (ii) the sequence does not initially satisfy condition (i) but can be shifted cyclically until condition (i) is satisfied.

For example, {1, 3, 5, 6, 7, 9, 4, 2} is a bitonic sequence as it satisfies condition (i). Similarly, the sequence {7, 8, 6, 4, 3, 1, 2, 5}, which does not satisfy condition (i), is also bitonic as it can be shifted cyclically to obtain {2, 5, 7, 8, 6, 4, 3, 1}.

Theorem II.6.a.1: [A1] Let $\{a_1, a_2, ..., a_{2n}\}$ be a bitonic sequence. If $d_i = \min(a_i, a_{n+i})$ and $e_i = \max(a_i, a_{n+i})$ for $i = 1, ..., n$, then

(i) $\{d_1, d_2, ..., d_n\}$ and $\{e_1, e_2, ..., e_n\}$ are each bitonic, and
(ii) $\max(d_1, d_2, ..., d_n) \le \min(e_1, e_2, ..., e_n)$.

The proof is given in [A1]. This theorem implies that we can sort a bitonic sequence $\{a_1, a_2, ..., a_{2n}\}$ into increasing order as follows:

(1) Using n comparators the two subsequences

$$\min(a_1, a_{n+1}), \min(a_2, a_{n+2}), ..., \min(a_n, a_{2n})$$

and

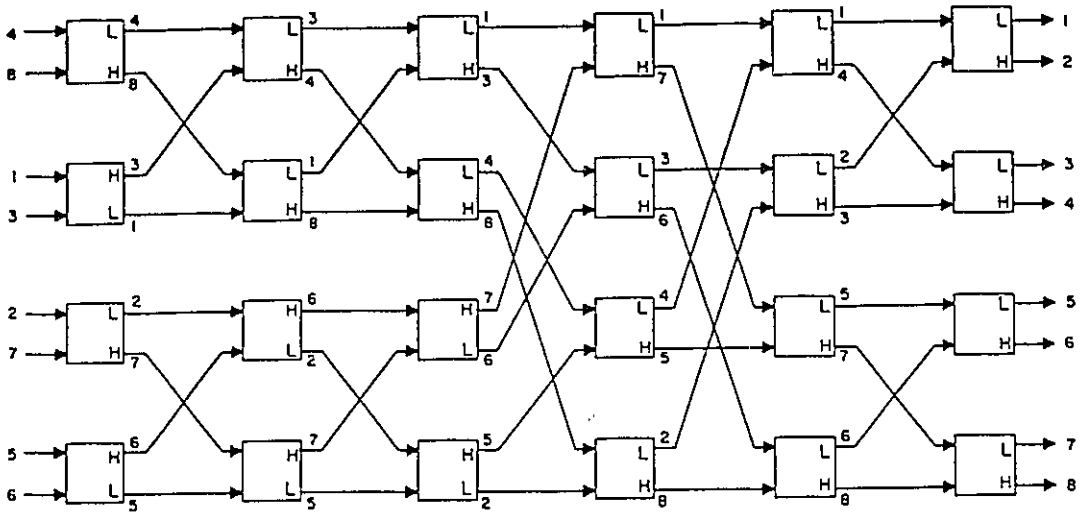
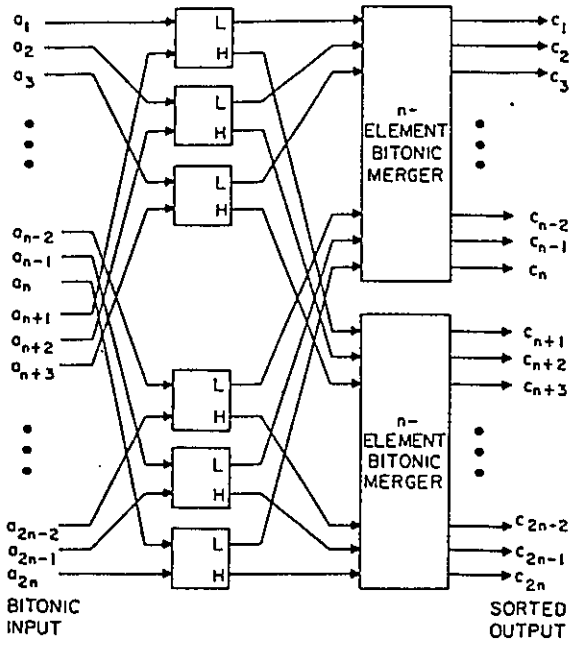
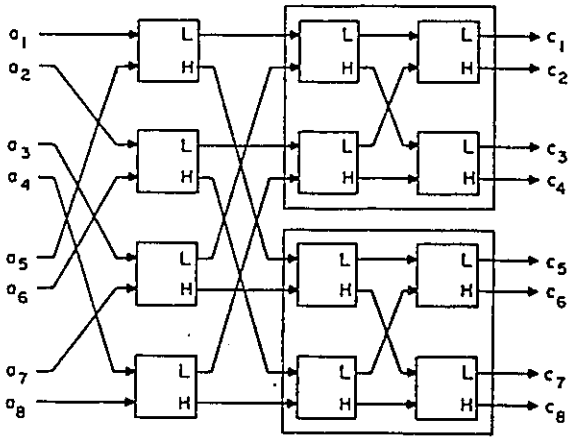


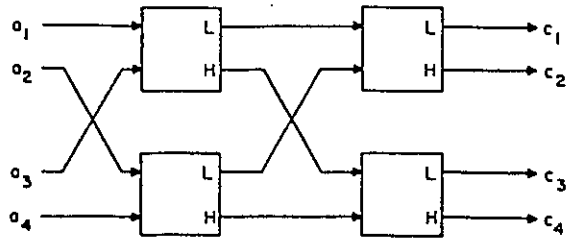
Fig. 1d Sorting (4, 8, 1, 3, 2, 7, 5, 6) by Bitonic Sort.



Bitonic Merger.



Bitonic Merger for a sequence of length 8.



Bitonic Merger for a sequence of length 4.

$$\max(a_1, a_{n+1}), \max(a_2, a_{n+2}), \ldots, \max(a_n, a_{2n})$$

are created

(2) Each of these two subsequences being bitonic it can be sorted recursively using a sorter for bitonic sequences of length n. Since no element of the first subsequence is larger than any element of the second subsequence, the n smallest elements of the full sorted sequence will be produced by one of these sorters and the n largest elements by the other one. Such a network is also known as Bitonic Merger or Batcher's sorting network.

If an arbitrary sequence S of n elements in random order is to be sorted, then bitonic subsequences of S are sorted and combined to form larger bitonic subsequences until a bitonic sequence of length n is obtained, which is finally sorted. The algorithm is known as *Bitonic Sort*. It should be noted that the n elements to be sorted must be available and input to the network simultaneously.

A network for sorting the random sequence S = {4, 8, 1, 3, 2, 7, 5, 6} using Bitonic Sort is shown in figure 1d. Note:

(1) In order to produce the decreasing part of a bitonic sequence, some of the comparators invert their output lines and produce a pair of numbers in decreasing order.

(2) After the input goes through the first rank of comparators, two bitonic sequences each of length 4 are produced. Each of these is then fed into a Bitonic Merger for sequences of length 4 (the comparators in columns 2 and 3). This results in a single bitonic sequence of length 8, which is now sorted using Bitonic Merger for sequences of length 8 (the comparators in columns 4, 5, and 6).

Figure 2a shows a 64-input bitonic sorting network, which consists of the comparator elements as illustrated in figure 2b. When this network is implemented on a binary VLSI chip by using 8-bit comparators for fully parallelized sorting, the interconnection area covers almost 70 percent of the chip area due to the complex topology of interconnections. The layout is shown in figure 3.

Suppose that the maximum degree of multiplexing is 4, that is 4 binary comparators are mapped onto a single SVL comparator using 4 wavelength components. The number of binary

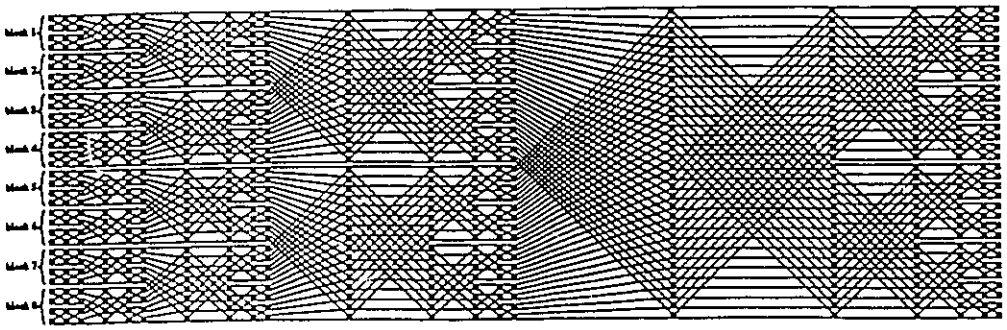
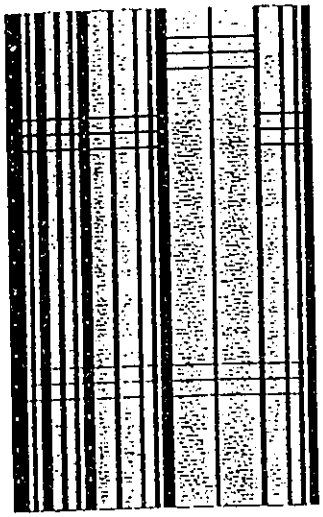


Figure 2 : 64-input Batcher's sorting network: (a)binary network; (b)sorting element; (c)SVL network.



: *Interconnection*

Figure 3 : Layout of 64-input bitonic sorting network built with 8-bit comparators.

Table 2: Comparison of binary and SVL configurations.

	Binary B	Set-valued S	Ratio S/B
Number of I/O lines	128	32	$\frac{1}{4}$
Number of modules	672	168	$\frac{1}{4}$
Number of interconnections	20768	5192	$\frac{1}{4}$
Interconnection area	$5632\lambda^2$	$376\lambda^2$	$\frac{1}{15}$

The maximum degree of multiplexing r : 4
Spacing between grid lines : λ

comparators being greater than 4, the sorting network is then regularly partitioned into 8 (= 32/4) blocks. Applying the multiplexing procedure to each block, the SVL network for multiplex sorting is derived as shown in figure 2c.

To evaluate the interconnection area based on a common measure, the authors adopt the idea of grid model [F1] for optical wavelength integrated circuit and VLSI circuit, that is, the interconnection area is proportional to the number of modules and the number of interconnections which cut across the area between modules; areas are expressed in terms of the number of squares of the grid (along whose lines wires run) covered by a circuit. Under the ideal assumption that one SVL module occupies the same area as does one binary module, the interconnection area among the modules is 1/15 of corresponding binary network. This ratio approaches $1/r^2 = 1/16$ as the problem size grows. The comparison of the SVL sorting network realized on an optical wavelength integrated circuit and the binary VLSI sorting network is shown in table 2.

In [A3], there is a detailed analysis of the impact of the biological molecular interconnection-free computing on the hardware cost of sorting networks. It focuses on the bubble sorting network which is a locally interconnected network (figure 4), and the bitonic sorting network which is a globally interconnected network. As shown in figure 5, we can see that, for N inputs, the bitonic sorting network requires $O(\lg^2 N)$ comparison steps, and therefore is much faster than the O(N) steps bubble sorting network. The authors estimate the total area for each sorting network as:

$$A_{Bubble} = A_{Bitonic} = O(N^2 \lg^2 N)$$

The number of comparator modules in each sorting network is:

$$M_{Bitonic} = (1/4)N(\lg N)(\lg N + 1) = O(N \lg^2 N)$$
$$M_{Bubble} = (1/2)N(N + 1) = O(N^2)$$

Thus, we can see that the area in the bubble sorting network is due to the large number of comparator modules used, while the area in the bitonic sorting network is dominated by complicated wire area (interconnections). Figure 6 plots the VLSI area versus the problem size N for the two types of sorting networks.

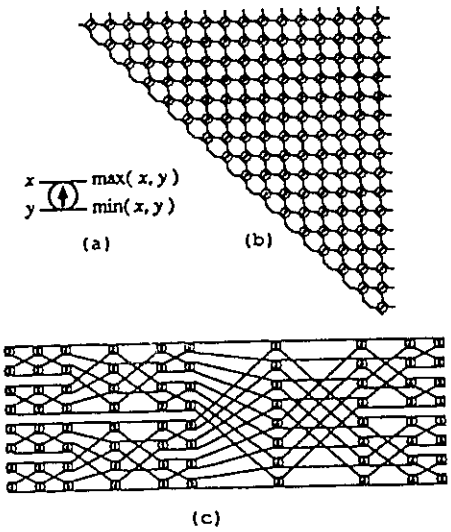


Figure 4: Sorting networks: (a)Comparator module, (b)16-input bubble sorting network, and (c)16-input bitonic sorting network.

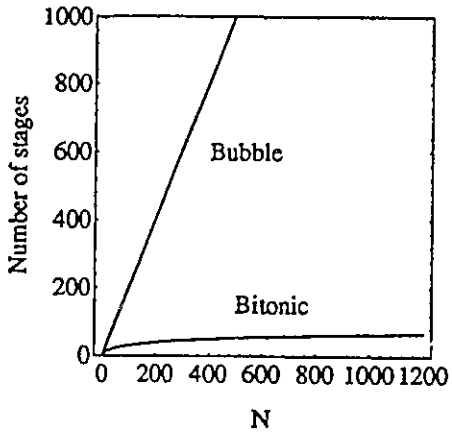


Figure 5: Comparison of sorting networks with respect to the number of comparator stages.

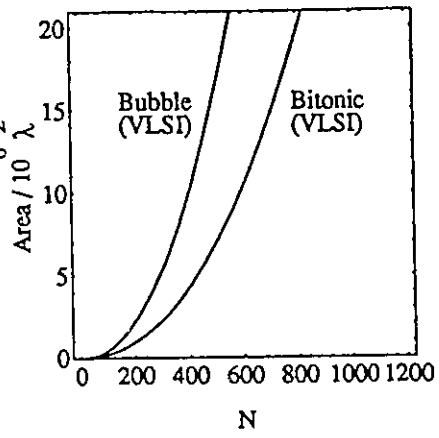


Figure 6: Comparison of areas of sorting networks implemented on VLSI chips.

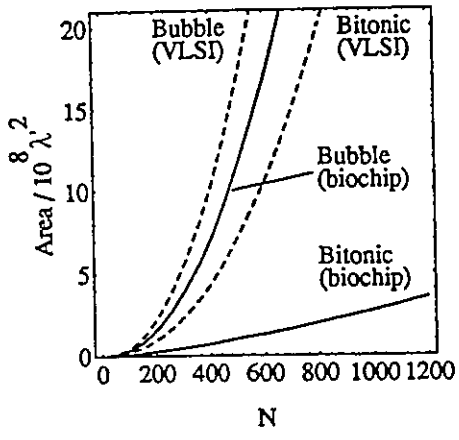


Figure 7: Comparison of areas of sorting networks implemented on biochips.

Arbitrary binary logic circuits can be simulated in biological molecular devices (bio-devices) networks using the isomorphic transformation. Hence, the two sorting networks can be realized with bio-devices networks. The communication links among binary modules can be realized based on parallel distribution and parallel selection of biological molecules assigning a specific substrate (elementary carrier) to each link. The estimated total area of each of the SVL-based sorting network thus obtained is:

$$A_{SVL\text{-}bubble} = O(N^2 \lg N)$$
$$A_{SVL\text{-}bitonic} = O(N \lg^3 N)$$

Figure 7 plots the biochip and VLSI chip area versus problem size N for the two types of sorting networks, assuming that a biochip comparator occupies the same area as a VLSI comparator. As this figure demonstrates, regular and locally interconnected networks like the bubble sorting network may be better or effectively implemented on VLSI chips rather than on biochips. In contrast, for the bitonic sorting network, bio-devices occupies only $O(N \lg^3 N)$ area, while the VLSI implementation needs $O(N^2 \lg^2 N)$. This implies that biological molecular computing is very effective for reducing hardware requirements for highly parallel sorting network.

II.6.b. SVL-based parallel image processing

SVL network is essentially suitable for applications which require complex multiple input/output lines. Such structure is mostly found in parallel processors for real-time applications, such as real-time image processing. As an example, let's consider the design of an image processing system based on the 3×3 near-neighbor logical operation which is generalized by the following template matching. A template is a 3×3 binary image to be compared to a section of a binary input image. Let the binary input image be the n×m matrix (x_{ik}), let the binary output image be the n×m matrix (y_{ik}), and let the template be any 3×3 matrix. When an input pixel x_{ik} of the input image and its near-neighbor pixels x_{i-1k-1}, ..., x_{i+1k+1} are equal to the template pixels t_0, ..., t_8 respectively, then the output y_{ik} is equal to a given b. The value of b is, in other words, the next value of the center pixel only if there is a match. Otherwise, $y_{ik} = x_{ik}$. In the case of recursive near-neighbor operation, y_{ik} becomes a new input x'_{ik}. This function can be formulated as:

$$y_{ik} = f(x_{i-1k-1}, ..., x_{ik}, ..., x_{i+1k+1})$$

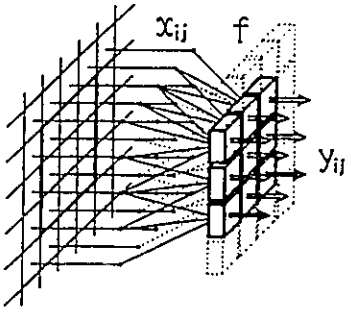


Fig.8 : Binary modules

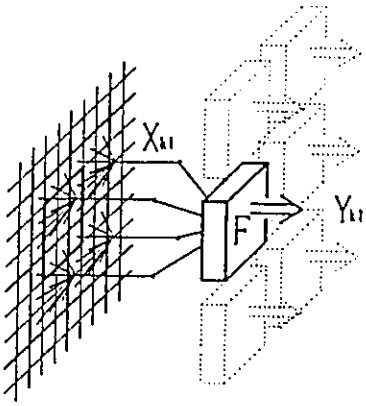


Fig. 9: Set logic module.

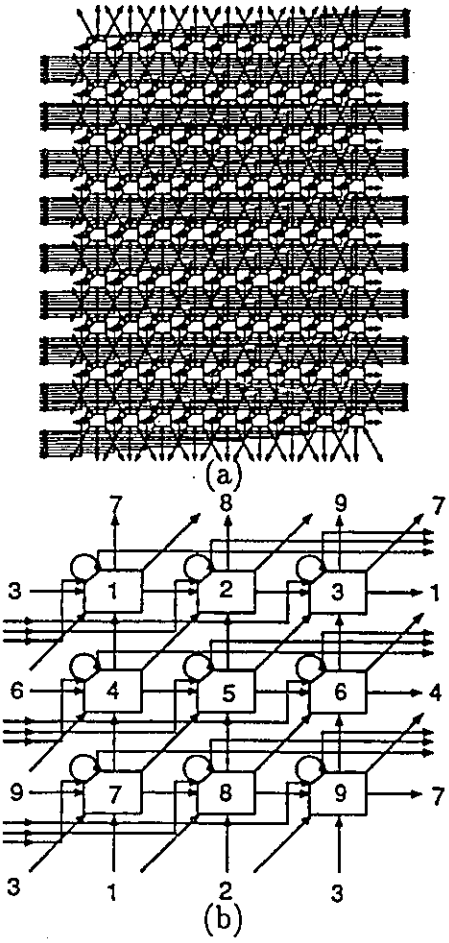


Figure 10: Comparison of binary and SVL image processor architectures. (a)binary architecture, (b)SVL architecture.

where i = 1, ..., m - 1, k = 1, ..., n - 1, and f denotes the binary template matching function. On the basis of this operation, various type of image processing, such as noise removal and contour extraction, can be systematically executed. In the conventional binary parallel image processing, a large number of modules are assembled to perform the function f for all pixels in parallel as illustrated in figure 8. For a 9×9 near-neighbor operation, this leads to the binary image processor architecture shown in figure 10a.

On the other hand, using frequency multiplexing with 9 as the maximum degree of multiplexing, nine binary modules can be multiplexed into a one common SVL module. This SVL module performs simultaneously nine binary functions f. Figure 9 illustrates the SVL module, while figure 10b shows the SVL image processor architecture for a 9×9 near-neighbor logical operation. Table 3 compares the two architectures [A2]. Table 4 shows the comparison for a recursive near-neighbor operation [Y2]. We see that there is a drastic reduction of wiring complexity which is almost proportional to 1/9. Interconnection-free parallel image processing architectures can also be designed using bio-devices [A5].

II.6.c. Other possible applications

SVL systems are very effective for reducing hardware requirements for highly parallel architectures. Consequently, any highly conventional parallel architecture with global interconnection between modules can be converted into a SVL architecture by simply multiplexing the elementary carriers (logic values).

II.7. Conclusion

In this chapter, we have described the concept of multiplex computing for superchips with massive parallelism, and discussed some possibilities of implementation and application. Nevertheless, the key to success lies in developing new devices and hardware algorithms inherently well suited for multiplex computing. Another thing is that SVL systems are more and very effective for reducing the hardware requirements for globally interconnected architectures. Locally interconnected architectures may be effectively implemented on VLSI chips rather than SVL chips. Finally, even if the SVL devices are very slow in comparison to VLSI devices, their main advantages is their massive parallelism. This massive parallelism compensates largely the slow switching speed particularly when the degree of multiplexing is large.

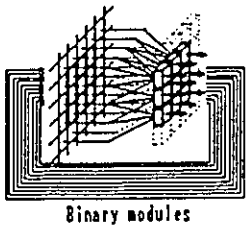
Table 3: Evaluation

Scale	Number of I/O lines		Number of interconnections among modules	
	Binary	Set-valued	Binary	Set-valued
9 × 9	81	9	729	36
⋮				
99 × 99	9,801	1,089	88,209	4,356
n × n	n^2	$\frac{1}{9}n^2$	$9n^2$	$\frac{4}{9}n^2$

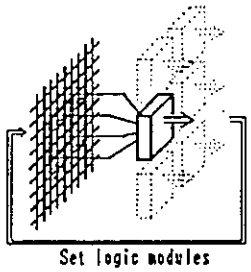
Table 4: Evaluation.

Scale	Number of I/O lines		Total interconnection for modules		Number of Transistors	
	Binary	Set	Binary	Set	Binary	Set
3 × 3	25	8	81	8	558	2,016
9 × 9	121	32	729	72	5,022	18,144
⋮						
99 × 99	10,201	2,312	88,209	8,712	6.1×10^5	2.2×10^6
n × n	$(n+2)^2$	$\frac{2}{9}(n+3)^2$	$9n^2$	$\frac{8}{9}n^2$	$62 \times n^2$	$224 \times n^2$

(Including adders)



Binary modules



Set logic modules

CHAPTER III
A PROMINENT EXAMPLE OF CARRIER COMPUTING: BIOLOGICAL MOLECULAR COMPUTING

III.1. Introduction

Over the past two decades, VLSI circuit technology has advanced rapidly, dramatically improving both performance and cost per function of integrated circuits. Processing and design methodologies have evolved significantly, leading to increased chip density and complexity. Unfortunately, in complex VLSI systems, these increases cause serious interconnection problems in chip area, power consumption, and noise. Moreover, interconnection complexity sometimes restricts the entire VLSI architecture within narrow limits of modularity, regularity, local interconnection, and minimized input/output [K3,S2].

One promising candidate for breaking through these difficulties is the biological molecular computer: a computer based on the dynamism of biological molecular activities, rather than on electronic switching [C1]. [A5,A7] have modeled primitive biomolecular switching devices that select and distribute specific biomolecules to perform highly parallel interconnection-free computing. The model is based on the specificity of enzymes -the biological catalysts- in their choice of reactants, called substrates. It represents the parallel distribution of logical information in the varieties of substrates molecules and uses enzyme specificity for parallel selection.

In a biomolecular processor based on this model, the specified substrates are broadcast in solution from sources. They diffuse with random molecular motion and carry information by their presence and absence in solution. At the specified destination, enzyme-based biosensors selectively detect the released substrates, which automatically triggers a specific biomolecular switch in the solution. In this sense, the data transmission is *interconnection-free*.

The set logic algebra (SLA) is the foundation of the bio-computing system. The varieties of substrates molecules represent the SVL logic values. Furthermore, since typical enzyme molecules might consist of 300 amino acids chosen from 20 types that commonly occur in living systems [C1], the system can also use the enormous number of potential enzymes shapes for discriminating logic values

Electronic VLSI systems are very effective for executing fast iterative arithmetic computations. Because biomolecular computers have low data rates, they are not meant to compete with electronic VLSI systems in this area. Instead, their advantages is massive and natural parallelism. They offer a new parallel processing architecture that is not restricted to the conventional interconnected structure.

This chapter describes the basic concept of interconnection-free biomolecular computing and discusses how it might be realized at the present state of technology.

III.2. Enzymes and specificity

Enzymes are catalysts: they speed up the rates of chemical reactions without undergoing any permanent change themselves. Most enzymes are highly specific both in their choice of reactants (called substrates) and in the reactions they catalyze. An enzyme recognizes specific substrate molecules in the fashion of a key fitting into a lock. The enzymes perform the highly selective detection almost independently of each other. This attractive feature of *parallel selection* is essential in the interconnection-free molecular computing.

Each reaction in a living cell is catalyzed by its own particular enzyme, so there are many enzymes in a given cell. A precise estimate of their number is difficult to calculate, but it seems that a bacterial cell makes about 3,000 different proteins and a higher eukaryote cell makes about 50,000. The majority of these proteins are enzymes [P2].

In addition to these natural enzymes, it is now possible to make new protein catalysts, such as catalytic antibodies [L1], to serve as enzymes. This new technology relies on the great diversity of the immune system, which can make perhaps 100 million different antibodies.

III.3. Biomolecular switching devices

An important characteristic of every enzyme is its specificity. The special spatial configuration of the enzyme exactly fits the appropriate region of the substrate somewhat in the fashion of key fitting into a lock [P2]. This unique selectivity offers the possibility of realization of bio-switch devices. In this section we describe the functioning of bio-devices presented in [A4,A5,A7].

Let the set of enzymes be given by $L_e = \{e_0, e_1, ..., e_{r-1}\}$. Each enzyme e_i in this set has the corresponding substrate s_i, and let the set of the substrates be $L_s = \{s_0, s_1, ..., s_{r-1}\}$. The specific reaction between the enzyme e_i and the substrate s_i makes a new product p_i. Let the set of the products thus obtained be $L_p = \{p_0, p_1, ..., p_{r-1}\}$. In the following discussion, let the r logic values correspond to the r substrates and enzymes as $e_0,s_0 \equiv 0$, $e_1,s_1 \equiv 1$, ..., $e_{r-1},s_{r-1} \equiv r - 1$. That is, we assume that L_s and L_e are equivalent to the logic value set L as $L_e \equiv L_s \equiv L = \{0, 1, ..., r - 1\}$. The logic values thus represented can be transmitted simultaneously in solution. This simultaneous transmission is interpreted algebraically as *logic value multiplexing*. Furthermore, an enzyme-based biosensor [Turner] can exactly discriminate the molecular information. Such a sensor consists both of detector enzymes that are immobilized on a membrane and of a transducer that produces an electric signal in response to the enzyme action.

Three types of biomolecular switching devices have been proposed in [A7] as the basic components of the molecular computing system: the bio-pass (BP) gate, the bio-output (BO) gate and the bio-complement (BC) gate.

The bio-pass gate performs the logic values selection or discrimination function based on the enzyme-substrate reactions. Let $S \subseteq L_s$ and $E \subseteq L_e$. The bio-pass gate is defined as

$$BP(S; E) = \begin{cases} \varnothing_L & \text{if } S \subseteq E \\ S \setminus E & \text{otherwise} \end{cases}$$

where $\varnothing_L$ denotes the non existing condition of the substrates belonging to L, and where $\setminus$ denotes the set difference operation. The set of substrates S is applied to the input membrane of the bio-pass gate. Let assume s_i to be an arbitrary element of S. If there exists the enzyme e_i in E (which is the set of enzymes immobilized on the internal membrane of the bio-pass gate), the enzyme-substrate reaction between e_i and s_i causes the chemical change of the substrate s_i: we say that the substrate s_i is captured by the enzyme e_i so that the other kinds of substrates pass through the output membrane. If e_i does not exist in E, then s_i pass through the output membrane as it is. That is, the output of the bio-pass gate becomes $\varnothing_L$ if and only if the inclusion relation $S \subseteq E$ is satisfied. Thus the BP gate behaves simply like a set difference operation.

The bio-output device generates the specified set of logic values. Let $S \subseteq L_s$ and $C \subseteq L_s \cup L_p$. The bio-output generator is defined as

$$BO(S;\ C) = \begin{cases} S & \text{if } C = \varnothing_L \\ \varnothing_L & \text{otherwise} \end{cases}$$

A biosensor composed of detector enzymes and electrochemical transducer causes the change of permeability of the output membrane. If the control input C contains at least one kind of substrate belonging to L_s, the biosensor detects this biochemical substance and its output signal inhibits the release of the substrates S *programmed* in the bio-output generator. Therefore, the substrate S is diffused to the output if and only if $C = \varnothing_L$.

The bio-complement gate performs the complement operation on the set of substrates applied to the input. Let $S \subseteq L_s$. The bio-complement gate is defined as

$$BC(S) = \overline{S} = L \setminus S$$

When the biosensor senses the substrate s_i in the input set, the diffusion of s_i is inhibited at the output stage. Remark: $BP(L_s; S) = BC(S)$.

[A4] have proposed a biomolecular switching device, the BIO gate, which performs the parallel selection and distribution function. Let X be the set of input substrates ($X \subseteq L_s$), and let $S \subseteq L_s$ and $E \subseteq L_e$. The BIO gate is defined as

$$BIO(X;\ E,\ S) = \begin{cases} S & \text{if } X \cap E = \varnothing_L \\ \varnothing_L & \text{otherwise} \end{cases}$$

The electronic signal of the biosensor controls the output of the substrates *programmed* in the device. On the surface of the biosensor, the set E of enzymes which responds selectively to a set of substrates is immobilized (hence the sensor is a *multiple-enzyme biosensor*). Only when the set of input substrates X in solution includes at least one kind of substrate which reacts to an enzyme in the set E (we say that the substrate is *captured* by the enzyme), the enzyme electrode detects this substance, and its electric signal inhibits the release of the substrates S. That is, the substrates are distributed if and only if $X \cap E = \varnothing$. This bio-device can simulate the bio-pass gate, the bio-output gate and the bio-complement gate.

III.4. Set logic network

In the proposed model, all the logical information is coded into the varieties of substrates and enzymes. Since these biochemical substances carry information by their presence and absence, the behavior of these substances can be essentially expressed in terms of set-theory. The set logic system, which can be constructed on the basis of Boolean algebras is the foundation of carrier computing systems including molecular computing system (see chapter I).

The basic operations in set logic are union $\cup$, intersection $\cap$ and complement $^-$. However, Boolean algebras by themselves do not suffice as a basis for molecular switching theory since they are not functionally complete (except for 2-valued logic case). To achieve functional completeness for values other than 2, [A7] introduced a specific unary operator called *set-theoretic literal*. Let X be a variable on 2^L, and let A, B $\in 2^L$. The set-theoretic literal denoted by ${}^A X^B$ is defined as

$$ {}^A X^B = \begin{cases} L & \text{if } A \subseteq X \subseteq B \\ \varnothing_L & \text{otherwise} \end{cases} $$

The set $\{\cup, \cap, {}^A X^B\}$ is complete with constants over 2^L. [A5] gives some useful properties of set-theoretic literal for the systematic synthesis of set logic functions, and [A7] shows that any set logic function of n variables can be written in the union-of-intersections form via appropriate simplification using the properties of the literal as follows

$$ F(X_1, ..., X_n) = \bigcup_{i=1}^{m} P_i \cap {}^{A_{i1}} X_1^{B_{i1}} \cap ... \cap {}^{A_{in}} X_n^{B_{in}} $$

where $P_i, A_{ij}, B_{ij} \in 2^L$, i = 1, 2, ..., m and j = 1, 2, ..., n.

For example, let L = {0, 1}. The function $(X_1 \setminus X_2) \cup (X_2 \setminus X_1)$ can be expressed as

$$ F(X_1, X_2) = (\{0\} \cap {}^{\{0\}} X_1^L \cap {}^{\varnothing} X_2^{\{1\}}) \cup (\{1\} \cap {}^{\{1\}} X_1^L \cap {}^{\varnothing} X_2^{\{0\}}) \cup (\{0\} \cap {}^{\varnothing} X_1^{\{1\}} \cap {}^{\{0\}} X_2^L) \cup (\{1\} \cap {}^{\varnothing} X_1^{\{0\}} \cap {}^{\{1\}} X_2^L). $$

The set logic functions $F(X_1, ..., X_n)$ can be realized with the bio-pass, bio-output and bio-complement gates and constructed as follows [A7]:

$$F(X_1, ..., X_n) = \bigcup_{i=1}^{m} BO(P_i; \bigcup_{j=1}^{n} [BP(BC(X_j); BC(A_{ij})) \cup BP(X_j; B_{ij})]).$$

Thus the set $\{\cup, BP, BO, BC\}$ is complete with constants over 2^L (Remark: in literature the $\cup$ operator is not included in the complete set, however, we must add it in the set because it appears in the general construction of the set-valued function F). This shows that any set logic function can be implemented by these three bio-devices and the union operation. The union operation is performed simply by mixing two groups of substrates in solution without any active devices. In order to reduce the complexity of the circuit, *mixed* union operations are fully used.

Also, [Y2] defines the set-difference (SD) gate, the set-generation (SG) gate and the union gate for frequency computing system. The definitions of the SD and SG gates are respectively equal to those of the BP and BO gates. The authors obtains exactly the same general construction of the set-valued function F (occurrences of BP, BO and BC are replaced respectively by SD, SG, and $^-$). Therefore the set $\{\cup, SD, SG, ^-\}$ is complete with constants over 2^L. Here again, the $^-$ operator is not included in their original complete set and we must add it for the same reason as mentioned above. Thus, in general, the set {$\cup$, BP or SD, BO or SG, BC or $^-$} seems to be the correct complete set system.

[A5] have shown that the set logic functions $F(X_1, ..., X_n)$ can also be implemented using only BIO gates. The BIO gate is represented algebraically as

$$BIO(X; E, S) = S \cap {}^{\varnothing}X^{\overline{E}}.$$

The authors transform the set logic functions into redundant set logic functions that can be always expressed by simpler literals, such as ${}^{\varnothing}X^{C}$. A redundant set logic function denoted by $\mathbf{F(X_1, ..., X_n)}$ is generally expressed using $F(X_1, ..., X_n)$ as follows:

$$\mathbf{F}(\mathbf{X_1}, ..., \mathbf{X_n}) = \bigcup_{i=1}^{m'} Q_i \cap {}^{\varnothing}\mathbf{X_1}^{D_{i1}} \cap ... \cap {}^{\varnothing}\mathbf{X_n}^{D_{in}}$$

$$= \bigcup_{i=1}^{m'} \mathrm{BIO}(\mathbf{X_1} \cup ... \cup \mathbf{X_n}; \bigcup_{j=1}^{n} \mathbf{X_j} \setminus \bigcup_{j=1}^{n} D_{ij}, Q_i)$$

where $\begin{cases} \mathbf{X_j} = X_j \cup f(\overline{X}_j) \\ \mathbf{F} = F \cup f(\overline{F}) \end{cases}$ and $f : L \Longrightarrow L'$, $L' = \{0', ..., (r-1)'\}$, $f(0) = 0', ..., f(r-1) = (r-1)'$

and $Q_i, D_{ij} \in 2^{\mathbf{L}}$, $\mathbf{L} = L \cup L'$, L' is a redundant logic value set.

Thus the set $\{\cup, \mathrm{BIO}\}$ is also complete with constants over $2^{\mathbf{L}}$. The network consists of this single type of bio-devices placed in buffer solution. The substrates molecules corresponding to SVL variables $\mathbf{X_1}, ..., \mathbf{X_n}$ are applied to the input and flow into the network simultaneously. Then specific bio-devices release the substrates which are mixed together to produce the final output of the function. That is, the bio-devices detect the input substrates in the buffer solution and control the enzyme reactions in parallel. The set logic network is thus composed of only separate devices, so the network topology need not to be specified in the design process.

III.5. Design procedure of biomolecular computing system

As it is stated in [A4,A5], the maximum parallelism of any given algorithm can be exploited through biological device networks (it can be extended to any carrier computing device networks) by converting its data flow graph (DFG) specification directly into SVL function. The DFG representation enables the exploitation of concurrency at the lowest possible level by treating each operation as an independent activity. In this way, extremely fine-grained parallelism for any arbitrary algorithm can be achieved, even if the algorithm itself does not exhibit high regularity. The design procedure for a biological molecular computing system is summarized as follows:

a) : *Representation of the algorithm by data flow graph.* The specification of computation is first represented in the form of a data flow graph whose nodes represent SVL functions and whose arcs represent the transmission of sets.

b) : *Expansion and simplification of SVL functions.* The complete set $\{\cup, \cap, {}^{A}X^{B}\}$ is used to express the node functions. Then the expression is minimized to reduce the number of required biological devices.

c) : *Design of redundant set logic function.* If a function contains a literal in the form of ${}^{A}X^{B}$ ($A \neq \varnothing$), such function must be transformed into a redundant set logic function.

d) : *Elementary carrier assignment.* (The elementary carriers here are substrate molecules). In the proposed system, all the logical information is transmitted by parallel distribution and parallel selection of substrate molecules. Hence, specific elementary carriers are assigned to logic values transmitted through the arcs (of the DFG), and all the functions are combined into a single SVL expression.

e) : *Design of the SVL network.* Since the SVL expression thus obtained have all the functional and topological information of the given DFG, the corresponding SVL network is now designed by converting the SVL expression into SVL circuit using the BIO gates.

III.6. Conclusion

It is difficult to discuss advantages and disadvantages formally at this early stage, but they may be summarized as follows:

Advantages:

* A great variety of information carriers are available.
* Massive parallelism.
* Interconnection-free.

Disadvantages:

* Slow switching speed.
* Long-term instability of biological molecules.
* The bio-device operation has not been confirmed yet even at an experimental level.

The biotechnological infrastructure is not yet mature enough for biomolecular computer designer to order the required materials. The key to success lies in finding (1) the systematic method to create new enzymes suitable for molecular computing and (2) simple and stable mechanism for controlling enzymes activities.

CHAPTER IV
SURVEY ON SET LOGIC ALGEBRA

IV.1. Functional completeness in multiple-valued and set logic

Historically the completeness problem was first studied in P_2. Although several complete systems were known earlier, a general completeness criterion was given by [P1] in 1921. This criterion, which has been rediscovered many times since, is most naturally expressed in terms of P_k-maximal sets.

Theorem IV.1.1: [R3] (Completeness criterion). A subset $F \subseteq P_k$ is complete in P_k if and only if F is a subset of no P_k-maximal set.▪

This is, in some sense, the most general completeness criterion because it is defined in terms of F only and covers all cases. Thus the completeness problem is solved by the determination of all P_k-maximal sets.

To describe the P_k-maximal sets, we need the following essential concept of *function preserving a relation* [R3]. Let $h \geq 1$. An h-ary relation ρ on L_k is a subset of L_k^h (i.e. a set of h-tuples over L_k) whose elements are written as columns. Given h rows n-vectors $a_i = (a_{i1}, a_{i2}, ..., a_{in})$ (i = 1, 2, ..., h) we write $(a_1, a_2, ..., a_h)^T \in \rho$ to indicate that $(a_{1j}, a_{2j}, ..., a_{hj})^T \in \rho$ for all j = 1, 2, ..., n, where T denotes the transpose (this means that the $h \times n$ matrix with rows $a_1, a_2, ..., a_h$ has all columns in ρ). We say that a n-place $f \in P_k$ preserves ρ if $(f(a_1), f(a_2), ..., f(a_h))^T \in \rho$ whenever $(a_1, a_2, ..., a_h)^T \in \rho$. Then the set of functions preserving ρ is denoted by $Pol\rho = \{f \mid (a_1, a_2, ..., a_h)^T \in \rho \Rightarrow (f(a_1), f(a_2), ..., f(a_h))^T \in \rho\}$. We need the following definitions for relations.

Let p be a prime. A group $G = (L_k, +)$ is called p-elementary abelian if G is abelian and $px = x + ... + x$ (p times) is the zero of the group for every $x \in L_k$. It is well-known that p-elementary groups on L_k exist if and only if $k = p^r$ for $r \geq 1$.

An equivalence relation on L_k distinct from $\{\{0\}, \{1\}, ..., \{k - 1\}\}$ and $\{\{0, 1, ..., k - 1\}\}$ is said to be non-trivial.

Let $1 \leq h \leq k - 1$. An h-ary relation ρ on L_k is central if $\rho \neq L_k^h$ and there is $\emptyset \subset C \subset L_k$ such that (1) $a_0 \in C \Rightarrow (a_0, a_1, ..., a_{h-1}) \in \rho$ for every $a_1, ..., a_{h-1}$, (2) $(a_0, ..., a_{h-1}) \in \rho \Rightarrow$

$(a_{s(0)}, ..., a_{s(h-1)}) \in \rho$ for every permutation s of the set $\{0, 1, ..., h - 1\}$ and, (3) $a_0 = a_1 \Rightarrow (a_0, a_1, a_2, ..., a_{h-1}) \in \rho$ for every $a_2, ..., a_{h-1}$. The unary central relations are just the proper non-null subsets of L_k.

Let $3 \leq h \leq k$ and let $m \geq 1$. The family $T = \{\theta_0, ..., \theta_{m-1}\}$ of equivalence relations on L_k is h-regular if (1) θ_j has h equivalence classes ($j = 0, ..., m - 1$), (2) the intersection $\cap\varepsilon_j$ of arbitrary equivalence classes ε_j of θ_j ($j = 0, ..., m - 1$) is non empty. The relation determined by T is the relation λ_T of all $(a_0, ..., a_{h-1}) \in L_k^h$ having the property that for each $0 \leq j \leq m - 1$ at least two elements among $a_0, ..., a_{h-1}$ are equivalent in θ_j.

[K4] has observed that the P_k-maximal sets can be best described as the set of all functions preserving a relation. All P_k-maximal sets are of the form Polρ for some relation ρ. Their full description is given in [R3,R4]. They are grouped into six classes and [R3] formulated the following general completeness criterion.

Theorem IV.1.2: [R3] Each P_k-maximal set is of the form Polρ where ρ is one of the six following relations on L_k:

- R1 Every partial order on L_k with a least and a greatest element.
- R2 Every relation $\{(x, s(x)) \mid x \in L_k\}$ where s is a permutation of L_k with k/p cycles of the same length p ($k = pr$ and p prime).
- R3 Every quaternary relation $\{(a_1, a_2, a_3, a_4) \in L_k^4 \mid a_1 + a_2 = a_3 + a_4\}$ where $(L_k, +)$ is a p-elementary abelian group ($k = p^r$ and p prime).
- R4 Every non-trivial equivalence relation on L_k.
- R5 Every central relation on L_k.
- R6 Every relation λ_T determined by an h-regular family T of equivalence relations on L_k ($h \geq 3$).

A set $F \subseteq P_k$ is complete if and only if for every relation ρ described under R1-R6 above there exists an $f \in F$ not preserving ρ.▪

Let Ri(k) denotes the number of relations in the class Ri for $i = 1, 2, ..., 6$. Let $\pi(k) = R1(k) + ... + R6(k)$ denotes the total number of P_k-maximal sets (i.e. number of relations in P_k). [R4] found a rather complicated formula for $\pi(k)$ by a purely combinatorial argument and gave the following table (for $3 \leq k \leq 7$):

k	R1	R2	R3	R4	R5	R6	π(k)
2	*1*	*1*	*1*	*0*	*2*	*0*	*5*
3	3	1	1	3	9	1	18
4	18	3	1	13	40	7	82
5	190	6	6	50	355	36	643
6	3285	35	0	201	11490	171	15182
7	88851	120	120	875	7758205	813	7848984
8			*30*		*549758283980*		> *549758283980*

As k grows most of the relations belongs to the class R5, and $\pi(k)$ increases very rapidly.

We cannot achieve functional completeness in many-valued set logic (i.e. the case $r \geq 2$) since the Boolean algebra on 2^R, say $[2^R, \varnothing, R, \cup, \cap, \bar{}]$, is not functionally complete except for the case $r = 1$ (see section I.3). A variation on the definition of completeness is then the concept of *Boolean completeness* in which we allow the use of Boolean functions in the compositions of non Boolean functions from a given subset F of $P_k \setminus BF$. This will be the topic of the following section. In the sequel, we let S(n) denote the set of n-place functions of a given set S and assume $k = 2^r$.

IV.2. S-completeness: Boolean completeness in set logic

In [A3,A4,A5,A6,A7] the question of constructing all set logic functions using Boolean functions is studied. Boolean functions are those composed from the set $\{\cup, \cap, \bar{}, c_0, c_1, ..., c_{k-1}\}$, where c_i is the constant function i. Since the set is not complete (except for $k \leq 2$), some functions are added to the set of Boolean functions to form a complete set. Boolean set-valued functions (which are a small fraction of the collection of set-valued functions) are very important set-valued functions, because they can be realized using binary standard electronic circuitry. In these considerations Boolean functions are considered *cheap* elements in the design of set logic functions. Following these investigations, it is interesting to characterize all sets of functions which become complete when Boolean functions are added to them.

Let S be a set of cheap functions. In set logic, since we are more interested to determine the completeness criteria under some set of Boolean functions, we assume S to be a non empty subset of $C = \{\cup, \cap, \bar{}, \text{constants}\}$, where constants = $\{c_0, c_1, ..., c_{k-1}\}$. The closure of the set S is denoted by [S] and contains all functions of P_k composed from S. A maximal set F in P_k is

said to be S-maximal in P_k if $S \subseteq F$. A subset F of $P_k \setminus [S]$ is said to be S-complete in P_k if the set $S \cup F$ is complete in P_k. A S-complete set F in P_k is called a S-base of P_k if no proper subset of F is S-complete in P_k. The rank of a S-base is the number of its elements. A function $f \in P_k$ is S-Sheffer (or Sheffer with S functions) for P_k if $\{f\} \cup S$ is complete in P_k, or equivalently, if $\{f\}$ is a S-base (of rank 1) of P_k. For example, bio-output [A7], conversion [M1] and literal [A5] functions are respectively $\{\cup, \bar{}, \setminus\}$-Sheffer, $\{\cup, \bar{}\}$-Sheffer and $\{\cup, \cap\}$-Sheffer for P_k.

Theorem IV.2.1: [D2] (S-completeness theorem in a general form). A subset F of functions in P_k is S-complete in P_k if and only if F is contained in no S-maximal set in P_k.■

We saw in the previous section that there is an exponential (in k) number of P_k-maximal sets divided into six classes R1-R6. We refer to these classes also as partial order, self-dual, quasilinear, equivalence, central and predicate classes of maximal sets, respectively. To determine which of the $\pi(k)$ maximal sets are S-maximal sets, it is sufficient to check for each of them whether it contains all S functions. The general solution to the S-completeness problem is then to find the S-maximal sets.

In the search of S-maximal sets, the following theorem is very useful. Let a three-place majority function be any function f that satisfies the following property: $f(x, x, y) = f(x, y, x) = f(y, x, x)$ for any x and y. One such function is $f(x, y, z) = (x \cap y) \cup (x \cap z) \cup (y \cap z)$, which is a [B] function.

Theorem IV.2.2: [B1] If a closed set can be described as set of functions preserving a h-ary relation ρ and it contains a majority function then $h \leq 2$.■

Corollary IV.2.3: If $\{\cup, \cap\} \subseteq S$ then there is no S-maximal set in the quasilinear and predicate classes and no S-maximal set corresponding to h-ary central relations for $h \geq 3$.■

Proof: Suppose $\{\cup, \cap\} \subseteq S$. First, the majority function which is the Boolean function given by $f(x, y, z) = (x \cap y) \cup (x \cap z) \cup (y \cap z) \in [S]$. Second, the arities of the quasilinear, predicate and central relations are respectively greater or equal to 4, 3 and 1. From Theorem IV.2.2 it follows that there is no S-maximal set in the quasilinear and predicate classes (as their arities are greater than 2) and no S-maximal set from central relations with arity greater than two.■

Two cases of S-completeness has been studied in literature and known as the problem of C-completeness ($S = C = \{\cup, \cap, \bar{}, c_0, c_1, ..., c_{k-1}\}$) and B-completeness ($S = B = \{\cup, \cap, \bar{}\}$). The C-completeness, called *weak* completeness in [S3] and Boolean completeness in [S4] (with the same meaning), determine the completeness criteria in set logic under compositions with C functions while the B-completeness problem studied in [D2] determine the completeness criteria in set logic under compositions with B functions.

IV.3. C-completeness and B-completeness in set logic

In [S3] a full description of C-complete sets, C-bases and C-Sheffer functions in P_4 is given. In [S4] the general case of arbitrary r is studied and it is proved that there are $2^r - 2$ C-maximal sets in r-valued set logic, all defined by some equivalence relations. [D1] determines classes of functions (under compositions with C functions), and enumerated C-bases and C-Sheffer functions in P_8. [D2] presents the B-completeness criteria and proves that there are $Bl(r) + 2^r - 3$ B-maximal sets in r-valued set logic (where Bl(r) is a Bell number, i.e. the number of set partitions of the set R), defined by Bl(r) - 1 central relations and $2^r - 2$ equivalence relations. These are all known results on C-completeness and B-completeness. The full description of B-complete sets, B-bases and B-Sheffer functions in P_4 and P_8 is presented in chapter VI.

Next we present all known lemmas for C-maximal sets and B-maximal sets determination (without giving their proof).

Lemma IV.3.1: [S4] The complement function does not preserve any partial order in the class R1.

Lemma IV.3.2: [S4] The constant functions do not preserve any relation in the selfdual class R2.

Lemma IV.3.3: [D2] There is no B-maximal set in the selfdual class R2.

Lemma IV.3.4: [D2,S4] There is no C-maximal set and B-maximal set in the quasilinear class R3.

Lemma IV.3.5: [D2,S4] There are exactly k - 2 C-maximal sets and k - 2 B-maximal sets in the equivalence class R4.

Lemma IV.3.6: [S4] For each unary central relation in the class R5 there exists a constant function which does not preserve it.

Lemma IV.3.7: [S4] All maximal sets determined by h-ary central relations in R5 for $h \geq 2$ are not C-maximal in P_k.

Lemma IV.3.8: [D2] There are Bl(r) - 1 B-maximal sets defined by unary central relations from the class R5.

Lemma IV.3.9: [D2,S4] There is no C-maximal and B-maximal sets in the predicate class R6.

Theorem IV.3.10: [S4] For $k = 2^r$, P_k has exactly k - 2 C-maximal sets.

Theorem IV.3.11: [D2] For $k = 2^r$, P_k has exactly Bl(r) + k - 3 B-maximal sets.

Constant functions trivially satisfy any equivalence relation and thus the analysis of C-maximal sets from [S4] can be applied in the B-completeness criteria, meaning that a maximal set from equivalence relation is B-maximal if and only if it is C-maximal. So, B-maximal sets from equivalence relations are equivalent to C-maximal sets.

For convenience, we denote the equivalence relations by ε_i, $1 \leq i \leq k - 2$, and corresponding C-maximal (B-maximal) sets by $E_i = Pol\varepsilon_i$. The relation ε_i is defined as follows. One equivalence class of ε_i consists of all numbers j such that $x \subseteq j \subseteq i \cup x$, where $\subseteq$ is the set inclusion ($a \subseteq b$ iff $a_m \leq b_m$ for $0 \leq m \leq r - 1$). The element x is any such that $i \cap x = 0$. Suppose that i has t 0's and r - t 1's. Then x in the last definition must have 0 at any position where i has 1 (for example, if i = 3 = 011 then x = 0 = 000 or x = 4 = 100) and thus there are 2^t equivalence classes of ε_i each containing 2^{r-t} elements. For example, for r = 3 the six equivalence relations which are preserved by all [C] and [B] functions are the following:

$$\varepsilon_1 = \{\{0, 1\}, \{2, 3\}, \{4, 5\}, \{6, 7\}\}$$
$$\varepsilon_2 = \{\{0, 2\}, \{1, 3\}, \{4, 6\}, \{5, 7\}\}$$
$$\varepsilon_3 = \{\{0, 1, 2, 3\}, \{4, 5, 6, 7\}\}$$
$$\varepsilon_4 = \{\{0, 4\}, \{1, 5\}, \{2, 6\}, \{3, 7\}\}$$
$$\varepsilon_5 = \{\{0, 1, 4, 5\}, \{2, 3, 6, 7\}\}$$
$$\varepsilon_6 = \{\{0, 2, 4, 6\}, \{1, 3, 5, 7\}\}.$$

For r = 2 we have 2 equivalence relations

$$\varepsilon_1 = \{\{0, 1\}, \{2, 3\}\}$$
$$\varepsilon_2 = \{\{0, 2\}, \{1, 3\}\}.$$

If the elements of L_k are drawn as edges of a r-dimensional hypercube then the equivalence relations ε_i are all partitions of the hypercube into equal size subcubes. The subdivision is uniquely determined by the subcube containing 0, and the subcube containing 0 is uniquely determined by its largest element (between 1 and k - 2).

We denote the unary central relations by γ_j, $0 \leq j \leq Bl(r) - 2$, and the corresponding B-maximal sets by $C_j = Pol\gamma_j$. The relation γ_j is any subset of L_k having the following properties: if $a \in \gamma_j$ then $\bar{a} \in \gamma_j$; if $a, b \in \gamma_j$ then $a \cap b \in \gamma_j$. Each element of γ_j is a subset of basic r-values $R = \{e_0, e_1, ..., e_{r-1}\}$ where e_i is 0 or 1. Thus γ_j corresponds to a subalgebra of Boolean algebra on R. Every set partition of R defines one such subalgebra as follows. If e_a and e_b are in the same class of given set partition then they either both are 0 or both are 1 in each element of γ_j. γ_j contains all such elements (therefore 0 and k - 1 in all cases). Every set partition (except the trivial one when all elements are in the same set) defines one subset γ_j corresponding to a B-maximal set. The number of elements in γ_j is 2^m where m is the number of equivalence classes in the set partition, $1 \leq m \leq r - 1$. For convenience, we let $\gamma_0 = \{0, k - 1\}$ denote the first unary central relation and also let $C_0 = Pol\gamma_0$ be the corresponding B-maximal set. For example, for r = 3 there exists four unary central relations which are preserved by all [B] functions:

$$\gamma_0 = \{0, 7\}$$
$$\gamma_1 = \{0, 1, 6, 7\}$$
$$\gamma_2 = \{0, 2, 5, 7\}$$
$$\gamma_3 = \{0, 3, 4, 7\}.$$

For r = 2 we have only one central relation

$$\gamma_0 = \{0, 3\}.$$

In the case of γ_3 we may use the index 4 (i.e. γ_4) for convenience instead of the index 3 (see the text following Lemma VI.3.1, section VI.3).

Now, the C-completeness and B-completeness criteria follow.

Corollary IV.3.12: [S4] A subset $F \subseteq P_k \setminus [C]$ is C-complete in P_k if and only if $F \setminus E_i \neq \varnothing$ for $1 \leq i \leq k - 2$.

Corollary IV.3.13: A subset $F \subseteq P_k \setminus [B]$ is B-complete in P_k if and only if $F \setminus C_i \neq \varnothing$ for $1 \leq i \leq Bl(r) + k - 3$.

Corollary IV.3.14: [S4] A function f is C-Sheffer in P_k if and only if $f \notin E_i$ for $1 \leq i \leq k - 2$.

Corollary IV.3.15: A function f is B-Sheffer in P_k if and only if $f \notin C_i$ for $1 \leq i \leq Bl(r) + k - 3$.

IV.4. Enumeration of functions and intersection properties of S-maximal sets in set logic

Once the completeness criteria are known, the intersection properties of S-maximal sets usually determine the S-complete sets, the S-bases and the S-Sheffer functions. The intersection properties of C-maximal sets are given here.

Theorem IV.4.1: [D1] $|E_i(n)| = 2^{(r-t)2^{rn}} + t2^{tn}$.

Lemma IV.4.2: [D1] $|\cap E_{2^r - 1 - 2^i}(n)| = 2^{r2^n}$, for $n \geq 1$ and $i = 0, 1, ..., r - 1$.

Lemma IV.4.3: [D1] $\cap E_{2^r - 1 - 2^i} = BF$, $i = 0, 1, ..., r - 1$.

Lemma IV.4.4: [D1] $E_i \cap E_j \subseteq E_{i \cup j}$.

Lemma IV.4.5: [D1] $\cap E_{2^i} = BF$, $i = 0, 1, ..., r - 1$.

Lemma IV.4.6: [D1] $E_i \cap E_j \subseteq E_{i \cap j}$.

The intersection properties of B-maximal sets left as open problem are discussed in section VI.3.

IV.5. Bio-algebras

[R2] introduced a class of algebras, called bio-algebras, which represent an extension of Boolean rings. The operations of these algebras are inspired by the works on biological computing. The authors defined and axiomatized the bio-algebra and prove that the finite algebras that satisfy the system of proposed axioms are functionally complete and incorporate the expected properties of the bio-pass and the bio-output. Thus, bio-algebras are primal algebras that are capable of model computation done by bio-circuits: bio-pass (BP), bio-output (BO) and bio-complement (BC) (see section III.3).

If $R = \{0, ..., r - 1\}$ is the set of fundamental values of a r-valued set logic then every j, $0 \leq j \leq r - 1$ represents a pair substratum-enzyme (assuming that we deal with r distinct enzymes). The bio-circuits mentioned above operate on the power set 2^R. The bio-pass is a two-place function defined by $BP(X, Y) = X \setminus Y$, that is by the set difference. The bio-complement is a unary function given by $BC(X) = \overline{X} = R \setminus X$. The bio-output is a two-place function given by

$$BO(X, Y) = \begin{cases} X & \text{if } Y = \varnothing \\ \varnothing & \text{otherwise} \end{cases}$$

The bio-pass and bio-complement functions are Boolean set logic functions while the bio-output is non Boolean. In order to obtain a natural algebra for this type of operations [R2] considers a Boolean ring $\mathcal{R} = (I, 0, 1, +, \cdot)$, that is a unitary, associative ring that consists of idempotent elements. Such a ring is commutative and of characteristic 2. In other words we have $x + x = 0$ and $x \cdot y = y \cdot x$ for every $x, y \in I$. [R2] defines the functions bp: $I^2 \rightarrow I$, bo: $I^2 \rightarrow I$ and bc: $I \rightarrow I$ by the equations

$$\begin{aligned} bp(x, y) &= x + xy \\ bo(x, y) &= \begin{cases} x & \text{if } y = 0 \\ 0 & \text{otherwise} \end{cases} \\ bc(x) &= 1 + x \end{aligned}$$

These functions are refered to as the bio-pass, the bio-output and the bio-complement functions on the Boolean ring $\mathcal{R}$.

Here we remind the reader the definition of the polynomials for an arbitrary ring. The set $Pol_n(I)$ of polynomials in n variables over a ring $(I, 0, 1, +, \cdot)$ is determined by the following rules:

i) for every $a \in I$ the constant function $f_a(x_1, ..., x_n) = a$ for every $x_1, ..., x_n \in I$ is in $Pol_n(I)$;

ii) for every i, $1 \leq i \leq n$ the projection function $p_i(x_1, ..., x_n) = x_i$ is in $Pol_n(I)$;

iii) if $f, g \in Pol_n(I)$ then the functions $f + g$ and fg defined by

$$(f + g)(x_1, ..., x_n) = f(x_1, ..., x_n) + g(x_1, ..., x_n) \text{ and}$$

$$fg(x_1, ..., x_n) = f(x_1, ..., x_n) \cdot g(x_1, ..., x_n),$$

for every $x_1, ..., x_n \in I$, are in $Pol_n(I)$.

[R7] (pp.37) gives the following useful theorem which characterize a polynomial of a ring.

Theorem IV.5.1: [R7] Let $(I, 0, 1, +, \cdot)$ be a Boolean ring. A mapping $b: I^n \rightarrow I$ is a polynomial if and only if it can be written in the canonical polynomial form

$$b(x_1, ..., x_n) = a_0 + \sum_{i_1...i_m}^{1, ..., n} a_{i_1...i_m} x_{i_1} ... x_{i_m},$$

where a_0 and $a_{i_1...i_m}$ are constants of I while the sum is extended over all subsets $\{i_1, ..., i_m\}$ of m distinct indices from the set $\{1, ..., n\}$. The coefficients a_0 and $a_{i_1...i_m}$ are uniquely determined by b. For instance, $b(x_1, x_2) = a_0 + a_1x_1 + a_2x_2 + a_{12}x_1x_2$.

The immediate consequence of this theorem is that

Corollary IV.5.2: [R2] The bio-output over a Boolean ring $(B, 0, 1, +, \cdot)$ is not a polynomial of the ring.

[R2] gives the definition of a bio-algebra. A bio-algebra is an algebra $\mathcal{A} = (I, 0, 1, bp, bo)$, where I is a set referred to as the carrier of the algebra, 0, 1 are two zero-ary operations and bp, bo are two binary operations linked by the following axioms:

i) $bp(x, x) = 0$,

ii) $bp(x, 0) = x$,

iii) $bp(x, bp(1, y)) = bp(y, bp(1, x))$,

iv) $bp(x, bp(y, z)) = bp(1, bp(bp(1, bp(x, y)), bp(x, bp(1, z))))$,

v) $bp(bp(x, y), z) = bp(bp(x, z), y)$,

vi) $bo(x, 0) = x$,

vii) if $y \neq 0$ then $bo(1, y) = 0$,

viii) $bo(x, y) = bp(x, bp(1, bo(1, y)))$,

for every $x, y, z \in I$.

[R2] prove that the finite algebras that satisfy this system of axioms are functionally complete and incorporate the expected properties of the bio-pass and the bio-output. We will give the results here without the proofs. Let $\mathcal{A} = (I, 0, 1, bp, bo)$ be a bio-algebra.

Proposition IV.5.3: [R2] In any bio-algebra, the above axioms imply

$$bp(0, 0) = 0, \quad bp(0, 1) = 0,$$
$$bp(1, 0) = 1, \quad bp(1, 1) = 0.$$

and

$$bo(0, 0) = 0, \quad bo(0, 1) = 0,$$
$$bo(1, 0) = 1, \quad bo(1, 1) = 0.$$

Corollary IV.5.4: [R2] For every $x \in I$ we have $x = bp(1, bp(1, x))$.

Corollary IV.5.5: [R2] For every $x \in I$ we have $bp(x, 1) = bp(0, x) = 0$.

Theorem IV.5.6: [R2] For every $x \in I$ we have $bo(x, x) = 0$.

Consider the binary operations $x \wedge y = bp(x, bp(1, y))$ and $x \vee y = bp(1, bp(bp(1, x), y))$, for every $x, y \in I$.

Proposition IV.5.7: [R2] We have

$$bp(1, y_1 \vee y_2) = bp(1, y_1) \wedge bp(1, y_2),$$
$$bp(1, y1 \wedge y_2) = bp(1, y_1) \vee bp(1, y_2)$$

for every $y_1, y_2 \in I$.

Proposition IV.5.8: [R2] Define $\overline{x} = bp(1, x)$. The 6-tuple $(I, 0, 1, \vee, \wedge, \overline{\ })$ is a Boolean algebra.

Any bio-algebra (I, 0, 1, bp, bo) can be regarded as a Boolean algebra (I, 0, 1, $\vee$, $\wedge$, $^-$) equipped with a supplementary operation bo. Naturally, since every Boolean algebra can be regarded as a Boolean ring, that is an idempotent ring, it is possible to define the ring operations $x + y$ and $x \cdot y$ by $x + y = bp(bp(1, bp(bp(1, x), y)), bp(x, bp(1, y)))$ and $x \cdot y = bp(x, bp(1, y))$, for every $x, y \in I$. A finite algebra is functionally complete if every function on that algebra can be regarded as a polynomial of the algebra.

Proposition IV.5.9: [R2] Consider the bio-algebra $\mathcal{B} = (I, 0, 1, bp, bo)$. The function x^y defined by

$$x^a = \begin{cases} 1 & \text{if } x = a \\ 0 & \text{otherwise} \end{cases}$$

is a polynomial of $\mathcal{B}$.

Corollary IV.5.9: [R2] Every finite bio-algebra is functionally complete.

IV.6. Approximation properties in set logic

Set-valued Boolean functions are very important set logic functions because they can be realized using binary standard electronic circuitry. Therefore, it is significant from a practical point of view to be able to decide whether a set logic function is Boolean and, in the negative case, which is the most likely situation, to be able to approximate portions of the graph of non Boolean functions by graphs of Boolean functions. This offers the possibility of building hybrid circuits, that is, circuits that use binary electronic components and non-binary optical or biological components which correspond to Boolean and non Boolean functions, respectively. We refer the reader to section I.4.b for preliminary definitions on approximation theory in set logic.

IV.6.a. The equivalence of a set logic function

Results concerning Boolean approximation of set logic functions in [R1] seek to identify partitions of the graphs of set logic functions such that the blocks of the partitions give the maximal interpolation domains of these functions by Boolean set-valued functions. The authors determined an upper bound on the complexity of bio-circuits that realize set logic functions. This bound is based on an equivalence attached to a set-valued function such that the classes of

the quotient set of the definition domain with respect to such an equivalence coincide with the sets on which they can evaluate the function by computing a value of a Boolean function. [R1] contains a list of one-place set logic functions based on the number of maximal interpolation classes. It might be possible to obtain (S-)completeness criteria of sets of set logic functions based on parameters of the attached Boolean interpolation sets. That is, if a given set F contains non Boolean functions, we can first replace (i.e. approximate) all or some of them by Boolean functions (from corresponding interpolation sets) and then determine the (S-)completeness criteria of the new set F'. F' contains the Boolean approximations of functions from F. The problem here is that if we decide to approximate all non Boolean functions of F, the new set F' may not be complete as it contains only Boolean functions. The interesting question is how to achieve functional completeness or S-completeness for such sets.

We briefly remind the reader the definition of the equivalence relations $\sim_F$ and $\equiv$ on 2^R. Let $F, G \in P_k(1)$. We have $X \sim_F Y$ if $(F(X) \oplus F(W) \subseteq X \oplus W$ if and only if $F(Y) \oplus F(W) \subseteq Y \oplus W$ for every $W \in 2^R$); the equivalence class of X with respect to $\sim_F$ is denoted by $[X]_F$. We have $F \equiv G$ if the function $h \in P_k(1)$ given by $h(X) = F(X) \oplus G(X)$ is a Boolean function. $\oplus$ is the set-theoretic symmetric difference operation. We denote the equivalence class of F by $[F]_{\equiv}$.

Theorem IV.6.a.1: [R1] Let $F: 2^R \rightarrow 2^R$. For every set $X \in 2^R$ there exists a Boolean function $b_X: 2^R \rightarrow 2^R$ such that $F(Y) = b_X(Y)$ for every $Y \in [X]_F$.

Proposition IV.6.a.2: [R1] Let $F, G: 2^R \rightarrow 2^R$. If $F \equiv G$ then $[X]_F = [X]_G$ for every $X \in 2^R$.

The equivalence classes $[X]_F$ of the quotient set $2^R/\sim_F$ constitute the sets on which we can approximate any function F of $P_k(1)$ by Boolean functions. The set of equivalence classes $[F]_{\equiv}$ determines a partition of $P_k(1)$ where each class $[F]_{\equiv}$ of functions gives in fact the maximal interpolation domain of theses functions (from $[F]_{\equiv}$) by Boolean functions.

Consider a set logic function F and its corresponding carrier computing circuit C. The results in [R1] allow the design of hybrid set logic circuits (biological and digital, for instance) equivalent to C, in which the Boolean components of F (ex. BP, $\cup$, $\bar{}$, ...) are implemented with binary circuits while we retain carrier computing switching devices (ex. bio-devices) for computing the non Boolean components of F (ex. BO, Conversion, ...). This enables the reduction of the implementation complexity of F. We define the implementation complexity C(F) of F as the number of non Boolean functions needed to design the corresponding carrier computing circuit C that computes F. This makes sense in real combinatorial set logic circuits,

since Boolean functions are usually obtained with small cost. Considering the bio-computing devices [R1] measures the complexity C(F) as the least number of bio-outputs needed to implement F. In general we measure C(F) as the least number of non Boolean devices needed by a circuit that computes F. Therefore, in view of theorem IV.6.a.1 we have $C(F) \leq |2^R/\sim_F|$ [R1].

Using an algorithm, the authors give a complete catalog of all 256 functions of $P_4(1)$ grouped according to their common equivalence relations. Consider for instance the function F given by F(0) = 3, F(1) = 2, F(2) = 1 and F(3) = 2. Applying the definition of $\sim_F$, the classes of the equivalence relation $2^{\{e_0, e_1\}}/\sim_F$ are then {0, 1}, {2}, {3}. For such an equivalence relation, the algorithm finds all functions G of $P_4(1)$ such that $G \equiv F$, i.e. $\sim_G = \sim_F$. The authors obtain 9 common equivalence relations, each corresponding to a class of functions.

Relations	{{0 1 2 3}}	{{1 2} {0 3}}	{{1} {2} {0 3}}	{{0} {2} {1 3}}	{{0} {1} {2 3}}	{{1} {3} {0 2}}	{{0} {3} {1 2}}	{{0 1} {2} {3}}	{{0} {1} {2} {3}}
num. of funct.	16	16	32	16	16	16	32	16	96

IV.6.b. Characterization of Boolean collections

[T1] contains certain combinatorial results involving collections of subsets of a given set whose characteristic function is a Boolean set logic function. The authors introduced the notion of *Boolean collection* of sets and explore several combinatorial aspects of these collections. These collections of sets appears to play a role in approximation of non Boolean set logic functions by Boolean functions and, therefore, are relevant in the study of carrier computing circuits, where set-valued functions are used. These results are useful for characterizing certain testing conditions involving set logic functions.

Recall the definition of a Boolean collection. A non empty collection of sets $C \subseteq (2^R)^n$ is a Boolean collection if there exists a n-place Boolean set logic function f such that $C = C_f$ where $C_f = \{X \in (2^R)^n \mid f(X) = \emptyset\}$.

Theorem IV.6.b.1: [T1] A collection of sets $C \subseteq 2^R$ is Boolean if and only if there are two sets $A, B \in 2^R$ such that $A \subseteq B$ and $C = \{X \in 2^R \mid A \subseteq X \subseteq \overline{B \setminus A}\}$.

Corollary IV.6.b.2: [T1] The set interval $[P, Q] = \{X \in 2^R \mid P \subseteq X \subseteq Q\}$ is a Boolean collection, for every $P, Q \in 2^R$.

Corollary IV.6.b.3: [T1] There are non Boolean set logic functions having Boolean collections as some of their level sets.

One such example is the set logic function given by $f(x) = \begin{cases} R & \text{if } P \subseteq X \subseteq Q \\ \varnothing & \text{otherwise} \end{cases}$. This a non Boolean function, called set literal function, used frequently in the study of carrier computing circuits (see section III.4). There exists a bijection between Boolean collections on 2^R and pairs of sets (A, B) for which $A \subseteq B$.

Corollary IV.6.b.4: [T1] There exist 3^r Boolean collections in 2^R. (There are 4^r pairs of sets in 2^R and only 3^r determine Boolean collections).

Lemma IV.6.b.5: [T1] Let $A, B, C \in 2^R$. The following hold:

i) $A\overline{C} \subseteq B\overline{C}$ if and only if $A \setminus B \subseteq C$.

ii) $AC \subseteq BC$ if and only if $C \subseteq \overline{A \setminus B}$.

Lemma IV.6.b.6: [T1] Let $A_i \in 2^R$ for $1 \le i \le 2k$. We have $A_1 \subseteq A_2 \cup A_3 \cup \ldots \cup A_{2k}$ if and only if $A_1 \setminus A_3 \setminus \ldots \setminus A_{2k-1} \subseteq \overline{(A_1 \oplus A_2) \setminus (A_3 \oplus A_4) \setminus \ldots \setminus (A_{2k-1} \oplus A_{2k})}$.

Theorem IV.6.b.7: [T1] The collection $C = \{(X_1, X_2) \in (2^R)^2 \mid f(X_1, X_2) = \varnothing\}$, where $f(X_1, X_2) = A_0^2 \oplus A_1^2X_2 \oplus A_2^2X_1 \oplus A_3^2X_1X_2$, is non empty if and only if

$$A_0^2 \subseteq A_1^2 \cup A_2^2 \cup A_3^2$$

and consists of all pairs of sets (X_1, X_2) that satisfy

$$A_0^2 \setminus A_2^2 \subseteq X_2 \subseteq \overline{(A_0^2 \oplus A_1^2) \setminus (A_2^2 \oplus A_3^2)},$$

$$A_0^2 \oplus A_1^2X_2 \subseteq X_1 \subseteq \overline{(A_2^2 \oplus A_3^2X_2) \setminus (A_0^2 \oplus A_1^2X_2)}.$$

Theorem IV.6.b.8: [T1] The collection $C_f = \{X \in (2^R)^n \mid f(X) = \varnothing\}$, is non empty if and only if $A_0^n \subseteq \bigcup\{A_i^n \mid 1 \le i \le 2^n - 1\}$ and consists of all tuples $X = (X_1, \ldots, X_n)$ that satisfy

$$A_0^m \setminus A_2^m \setminus \ldots \setminus A_{2^m-2}^m \subseteq X_m \subseteq \overline{(A_0^m \oplus A_1^m) \setminus (A_2^m \oplus A_3^m) \setminus \ldots \setminus (A_{2^m-2}^m \oplus A_{2^m-1}^m)}$$

for $1 \leq m \leq n$, where $A_i^m = A_{2i}^{m+1} \oplus A_{2i+1}^{m+1} X_{m+1}$ for $1 \leq m \leq n - 1$.

IV.6.c. Enumeration of Boolean collections

Lemma IV.6.c.1: [T1] There are $(2^m - 1)^r$ collections $\{A_1, \ldots, A_m\}$ in 2^R such that $A_1 \subseteq A_2 \cup \ldots \cup A_m$.

Corollary IV.6.c.2: [T1] There are $(2^{2^n} - 1)^r$ collections $\{A_0^n, A_1^n, \ldots, A_{2^n-1}^n\}$ in $(2^R)^n$ such that $A_0^n \subseteq \bigcup\{A_i^n \mid 1 \leq i \leq 2^n - 1\}$.

Consider the set $B = \{0, 1\}$. Define the mapping κ: $2^R \times R \rightarrow B$ by $\kappa(X, i) = 1$ if $i \in X$ and $\kappa(X, i) = 0$, otherwise. For a collection of sets denoted by $\{X_1, \ldots, X_n\}$ denote $\kappa(X_k, i)$ by x_k^i for $1 \leq k \leq n$ and $1 \leq i \leq r$. Similarly $\kappa(Y, i)$ is denoted by y^i. Each Boolean set logic function f: $(2^R)^n \rightarrow 2^R$ can be represented by r Boolean functions f_i: $B^n \rightarrow B$ where $f_i(x_1^i, \ldots, x_n^i) = y^i$ for $1 \leq i \leq r$ (see section II.4 on isomorphism relationship). We denote this situation by $f = [f_1, \ldots, f_r]$.

Lemma IV.6.c.3: [T1] Let f: $(2^R)^n \rightarrow 2^R$ be a Boolean function and let f_i for $1 \leq i \leq r$ be r Boolean switching functions such that $f = [f_1, \ldots, f_r]$. We have $f(X_1, \ldots, X_n) = \varnothing$ if and only if $f_i(x_1^i, \ldots, x_n^i) = 0$ for $1 \leq i \leq r$.

Theorem IV.6.c.4: [T1] The number of Boolean set logic functions f: $(2^R)^n \rightarrow 2^R$ for which there exists a n-tuple $(X_1, \ldots, X_n) \in (2^R)^n$ such that $f(X_1, \ldots, X_n) = \varnothing$ is equal to $(2^{2^n} - 1)^r$.

Corollary IV.6.c.5: [T1] Let $e_i = |\{(x_1^i, \ldots, x_n^i) \mid f_i(x_1^i, \ldots, x_n^i) = 0\}|$ for $1 \leq i \leq r$. The number e of tuples $(X_1, \ldots, X_n) \in (2^R)^n$ such that $f(X_1, \ldots, X_n) = \varnothing$ is given by $e = e_1 \cdots e_r$, where $1 \leq e_i \leq 2^n$ for $1 \leq i \leq r$.

Corollary IV.6.c.6: [T1] There are exactly $(2^{2^n} - 1)^r$ non empty Boolean collections $C \subseteq (2^R)^n$. For each such collection C, the Boolean function f: $(2^R)^n \rightarrow 2^R$ such that $C = \{X \in (2^R)^n \mid f(X) = \varnothing\}$ is uniquely determined.

Consider a subset $C \subseteq (2^R)^n$. Let F be a n-place set logic function. It is interesting to examine the collections of sets $C_{F,f} \subseteq (2^R)^n$ for which a Boolean function f: $(2^R)^n \rightarrow 2^R$ can be found such that $F(X) = f(X)$ for every $X \in C$. In others words, the problem is to determine the portions of $(2^R)^n$ where the computation of the function F can be replaced by the computation of a Boolean function f.

Consider the graph G(F) of F defined as $G(F) = \{(X_1, ..., X_n, X_{n+1}) \in (2^R)^{n+1} \mid F(X_1, ..., X_n) = X_{n+1}\}$. Then G(F) is a Boolean collection determined by the Boolean function h given by $h(X_1, ..., X_n, X_{n+1}) = f(X_1, ..., X_n) \oplus X_{n+1}$, where $f(X_1, ..., X_n) = F(X_1, ..., X_n)$. It is clear that $C_{F,f}$ is a fragment of G(F).

The *maximal* Boolean collections contained in the graph of non Boolean functions give the maximal sets on which we can approximate these non Boolean functions by Boolean ones. Therefore (S-)completeness properties of non Boolean set logic functions based on the maximal Boolean collections contained by these graphs present a great interest for implementation of hybrid circuits (biological and digital, for example).

IV.7. Conclusion

In this chapter we have discussed the completeness and approximation properties of set logic. Particularly, the S-completeness criteria are very useful to determine subsets of P_k which are S-completes. The next chapter will focus on how to find these S-complete sets using the criteria and some combinatorial methods such as classification and enumeration.

CHAPTER V
ALGORITHMS FOR CLASSIFICATION OF FUNCTIONS AND ENUMERATION OF BASES (IN SET LOGIC)

V.1. Introduction

The functions in P_k may be classified by their membership in the (S-)maximal sets, leading to a natural classification of (S-)bases into equivalence classes (called aggregates). This chapter presents classifications of functions and enumerations of aggregates given m (S-)maximal sets. First we describe an algorithm for facilitating the classification of set logic functions based on the (S-)completeness criteria. Second, on the basis of a backtrack procedure for lexicographic enumeration of all subsets of a set of n elements we present algorithms for determining all (S-)bases consisting of functions from a given (S-)complete set in P_k and for enumeration of all classes of (S-)bases in P_k [S5]. We apply these algorithms for the determination of the C-bases and B-bases of P_4 and P_8 and the one-place C-Sheffer and B-Sheffer functions for P_4 and P_8; the computational results thus obtained (for S = C) coincide with theoretical results given in [D1,S3].

V.2. Classification and enumeration of n-place (set logic) functions

Let $S_1, ..., S_m$ be the m (S-)maximal sets determined by the (S-)completeness criteria. As mentioned in the previous chapter, a subset F of P_k is (S-)complete in P_k if and only if for each i, $1 \le i \le m$, $F \setminus S_i \neq \emptyset$, or in other words, there is $f \in F$ such that $f \notin S_i$. This leads to the following: define the map $\varphi: P_k \to \{0, 1\}^m$ by setting $\varphi(f) := s_1...s_m$ where $s_i = 0$ if $f \in S_i$ and $s_i = 1$ if $f \notin S_i$. We call $\varphi(f)$ the *characteristic vector* of f. We put $f \equiv g$ if $f, g \in P_k$ have the same characteristic vector, i.e. if $\varphi(f) = \varphi(g)$. Clearly $\equiv$ is an equivalence relation on P_k and so it partitions P_k into pairwise disjoint non empty sets called *(equivalence) classes*. Note that for $f \equiv g$ we have either $f, g \in S_i$ or $f, g \notin S_i$ for all $i = 1, ..., m$. We write AB for $A \cap B$, A^0 for A and A^1 for $\overline{A}$ (A, B subsets of P_k). Clearly each class is of the form $S_1^{s_1} S_2^{s_2} ... S_m^{s_m}$ where $s_1, ..., s_m \in \{0, 1\}$. We will see in the next section how we can discuss (S-)completeness properties in P_k in terms of these classes instead of individual functions. The study of characteristic vectors provides information on the intersection properties of families of (S-)maximal sets, that is we can find why a given class of functions is empty or not or what kind of functions it contains (for a non empty class). The characteristic vectors can also be applied to seek the set of classes of functions which makes a given incomplete set (S-)complete.

In general, the number of possible classes (characteristic vectors) of (set logic) functions is 2^m where m is the number of (S-)maximal sets in P_k. So, for large m (say $m \geq 8$) it is better to use a computer program to generate most classes and enumerate (or generate) the functions in each class, in order to find the intersection properties in a fast way and also to seek for complete sets ((S-)bases and (S-)Sheffer functions).

The algorithm for classification and enumeration of (set logic) functions is based on a well known lexicographic generation of variations and goes in the following manner (see the definition of lexicographic order in section V.3.a). We generate each function and determine the possible class of functions which contains the function. Consider a set of k^n points in lexicographic order. We represent the i-th function as the k^n-tuple $f_i = (v_{i0}, v_{i1}, ..., v_{ik^n-1})$ where $v_{ij} \in L_k$ is the value of f_i in the j-th n-tuple point $x_j = (x_{j1}, ..., x_{jn}) \in L_k^n$ for $j = 0, ..., k^n - 1$ and $i = 0, ..., k^{k^n} - 1$. A function is simply any variation of k^n out of the k elements of L_k (a variation of m out of n elements is any sequence $(x_1, x_2, ..., x_m)$ where $0 \leq x_i \leq n$ and $1 \leq i \leq m$). For instance, for k = 8 and n = 1, (7, 3, 2, 4, 0, 0, 1, 5) is a function but (0, 7, 1, 8, 5, 2, 6, 4) is not. The general algorithm for classification of n-place (set logic) functions in Pascal-like notation goes as follows:

```
BEGIN
    { read(r);
      k = 2^r;   {in set logic}      or      read(k);   {in multiple - valued logic}
    read(n);                                          {number of variables}
    read(m);                                          {number of (S-)maximal sets}
                                                      {2^m = number of possible classes}
    class[1] := 00...00, ..., class[2^m] := 11..11;   {m-length characteristic vectors array}
    read([R_1, ..., R_m]);                            {array of m relations corresponding to (S-)maximal sets}
    set counter[1], ..., counter[2^m] to 0;           {number of functions in each class c}
    set v[1],..., v[k^n] to 0 ;                       {the first function: (0, 0, ..., 0)}
    REPEAT
            Determine the class c that contains f;
            increment counter[c] by one;
            j := k^n;
            WHILE v[j] = k - 1 DO j := j - 1;
            v[j] := v[j] + 1;
            set v[j + 1], ..., v[k^n] to 0;
    UNTIL j = 0;
    Print_out (class[c], counter[c]) for c := 1, ..., 2^m;
END.
```

The (set logic) functions are then generated in lexicographic order and classified. The function f belongs to the class $c = s_1...s_m \in \{0, 1\}^m$ if for $t = 1, .., m$, $f \in S_t$ whenever $s_t = 0$ and $f \notin S_t$ whenever $s_t = 1$. The procedure to check if a function belongs to a given class depends on the definitions of the m (S-)maximal sets which fall into the six classes R1-R6 of P_k-maximal sets (see section IV.1). Here we describe an algorithm which checks the inclusion of a function in (S-)maximal sets from either equivalence relations (class R4) or unary central relations (class R5). One such relation is represented as an array of classes (we consider a unary central relation as an *equivalence relation* having a unique *equivalence class*). Consider an equivalence relation ε, then a n-place function f preserves ε if for any set e^n (where e is an equivalence class of ε and e^n is the set all n-tuples taken out of e) the values in the points of e^n are all from the same equivalence class of ε. The class containing the coordinates of a given point of e^n and the class containing the value in that point are called the equivalence class *domain* and the equivalence class *image* of the point, respectively. Thus, to check the inclusion in a C-maximal set or B-maximal set, we need two arrays of length k^n; one to store the index of a class domain (from the relation ε (array of classes)) which contains all the coordinates of a given point and another one to store the index of a class image which contains the value of a given point. For example, if $k = 8$, $n = 1$, $\varepsilon = \{\{0, 1\}, \{2, 3\}, \{4, 5\}, \{6, 7\}\}$ and $f = \{3, 0, 3, 2, 4, 7, 6, 5\}$ then the arrays of domains and images are [1, 1, 2, 2, 3, 3, *4, 4*] and [2, 1, 2, 2, 3, 4, *4, 3*]. Looking at these two arrays, we see immediately that f does not preserve ε because the domain 4, for instance, has two distinct images (4 and 3); points in the set $\{6, 7\}^1$ map to distinct classes, point (6) maps to the 4th class and point (7) maps to the 3rd class of ε. Call *dom* and *img* the arrays of domains and images, then $1 \le dom[i] \le |\varepsilon|$ and $1 \le img[i] \le |\varepsilon|$ for $1 \le i \le k^n$. Also the i-th point x in the lexicographic order of the points of L_k^n can be obtained by converting the number i to its base k representation then the coordinate x_{ij} of x is simply the decimal equivalent of the j-th cipher of that base k number starting from the left, $1 \le i \le k^n$ and $1 \le j \le n$. For instance, for $k = 4$ and $n = 2$ we obtain 16 (base 10) = 33 (base 4), and then $x_{16,1} = 3$ and $x_{16,2} = 3$, i.e. $x_{16} = (3, 3)$. Either we can construct a k^n-length array of points before the REPEAT loop in the above algorithm (to save time) or we can compute a point at each iteration of the second FOR loop in the algorithm below (to save space). The algorithm to determine the class which contains a given function goes as follows.

```
Determine the class c of a function f (given an array of relations corresponding to C-maximal or B-maximal sets)
BEGIN
    set bin[1], ..., bin[m] to '1';                          {a m-length binary characteristic vector}
    FOR s := 1 to m do
        determine img[1], ..., img[k^n] from f and the relation R_s;
        FOR i := 1 to k^n do
            determine dom[i] from f and the relation R_s;    {dom[i] = 0 if the coordinates of point[i] are in}
                                                             {distinct equivalence classes of R_s}
            IF dom[i] ≠ 0 THEN
                image[dom[i]] := img[i];
        preserve := true;
        i := 1;
        WHILE preserve = true and i ≤ k^n do
            IF dom[i] ≠ 0 THEN
                preserve := img[i] = image[dom[i]];          {check if the points of a set e^n map to the same image}
            i := i + 1;
        IF preserve = true THEN bin[s] := '0';
    c := (decimal equivalent of the characteristic vector bin) + 1;
    class[c] := bin;
END
```

Variations are generated in $O(k^{k^n})$ steps. However, the procedure which checks the inclusion of a function in the m (S-)maximal sets is in $O(m \cdot k^n)$ steps and therefore increases the time complexity of the algorithm. There are also two other alternatives. First, we can use a set of k^n embedded FOR loops with k iterations each to generate all the k^{k^n} variations. This algorithm have the same time complexity but requires to modify the program whenever the parameter k is changed. Second, we can use a base k conversion algorithm to generate each function. That is, if we convert the number i to its base k representation then the value v_{ij} of the i-th function f_i is simply the decimal equivalent of the j-th cipher of that base k number starting from the left. For example, for k = 4 and n = 2 we have 16 points in lexicographic order, that is (0, 0), (0, 1), ..., (3, 3); also we obtain 21070 (base 10) = 0000000011021032 (base 4), and therefore $v_{21070,0} = f_{21070}(0, 0) = 0$, $v_{21070,1} = f_{21070}(0, 1) = 0$, ..., $v_{21070,14} = f_{21070}(3, 2) = 3$, $v_{21070,15} = f_{21070}(3, 3) = 2$, i.e. $f_{21070} = (0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 2, 1, 0, 3, 2)$. This algorithm of $O(k^{k^n})$ steps is less efficient than the others because of the additional base conversion algorithm.

The complement function is Boolean. If it belongs to the set S and if the function $f_i = (v_{i0}, v_{i1}, ..., v_{ik^n-1})$ belongs to the class c, then the function $f_{\bar{i}} = (v_{\bar{i}0}, v_{\bar{i}1}, ..., v_{\bar{i}k^n-1}) = (\overline{v_{i0}}, \overline{v_{i1}}, ..., \overline{v_{ik^n-1}})$ belongs also to c, that is $f_i \equiv f_{\bar{i}}$, where $\bar{i} = k^{k^n} - 1 - i$, $\overline{v_{ij}} = k - 1 - v_{ij}$ for $i = 0, ..., k^{k^n} - 1$ and $j = 0, ..., k^n - 1$. For example for r = 2 and n = 1, if $\{\bar{\ }\} \subseteq S$ we have $f_{18} = (0, 1, 0, 2) \equiv f_{237} = (3, 2, 3, 1)$, i.e. $\varphi(f_{18}) = \varphi(f_{237})$. The consequence is that we can use a variable t to

keep the number of functions we have classified, increment the class functions counters by two instead of one, and finally write *UNTIL* $t = k^{k^n} / 2$ instead of *UNTIL* $j = 0$. We need only to classify the first $k^{k^n} / 2$ functions. Therefore the algorithms run faster.

We applied the variations generation algorithm for the classification and the enumeration of set logic functions under C functions compositions. For r = 2 we denote each possible class of functions by the binary characteristic vector e_1e_2 where $e_1 = 0$ iff $f \in E_1$ and $e_2 = 0$ iff $f \in E_2$, E_1 and E_2 being the two C-maximal sets of P_4 (see section IV.3). For r = 3 there are six C-maximal sets E_1, E_2, E_3, E_4, E_5 and E_6, and we represent each possible class of set logic functions by a vector $e_1e_2e_3e_4e_5e_6$ where $e_1 = 0$ iff $f \in E_1$, $e_2 = 0$ iff $f \in E_2$, $e_3 = 0$ iff $f \in E_4$, $e_4 = 0$ iff $f \in E_6$, $e_5 = 0$ iff $f \in E_5$ and $e_6 = 0$ iff $f \in E_3$. We obtained the following results for n = 1, r = 2 and r = 3 (we give a one-place example function for each non empty class).

r = 2

Ordinal numbers	Decimal classes	Binary classes	One-place funct.	Number of funct.
1	0	00	(2, 2, 0, 0)	16
2	1	01	(2, 2, 0, 1)	48
3	2	10	(2, 0, 0, 0)	48
4	3	11	(2, 0, 0, 1)	144

r = 3

Ordinal numbers	Decimal classes	Binary classes	One-place funct.	Number of funct.
1	0	000000	(4, 4, 4, 4, 0, 0, 0, 0)	64
2	10	001010	(4, 4, 4, 4, 0, 0, 2, 2)	192
3	12	001100	(4, 4, 4, 4, 0, 1, 0, 1)	192
4	14	001110	(4, 4, 4, 4, 0, 1, 2, 3)	576
5	17	010001	(4, 4, 0, 0, 0, 0, 0, 0)	192
6	20	010100	(4, 4, 4, 5, 0, 0, 0, 1)	192
7	21	010101	(4, 4, 0, 1, 0, 0, 0, 1)	576
8	27	011011	(4, 4, 0, 0, 0, 0, 2, 2)	576
9	28	011100	(4, 4, 4, 4, 0, 0, 0, 1)	3648
10	29	011101	(4, 4, 0, 0, 0, 0, 0, 1)	11520
11	30	011110	(4, 4, 4, 4, 0, 0, 2, 3)	11520
12	31	011111	(4, 4, 0, 0, 0, 0, 2, 3)	36288
13	33	100001	(4, 0, 4, 0, 0, 0, 0, 0)	192
14	34	100010	(4, 4, 4, 6, 0, 0, 0, 2)	192
15	35	100011	(4, 0, 4, 2, 0, 0, 0, 2)	576
16	42	101010	(4, 4, 4, 4, 0, 0, 0, 2)	3648
17	43	101011	(4, 0, 4, 0, 0, 0, 0, 2)	11520
18	45	101101	(4, 0, 4, 0, 0, 1, 0, 1)	576
19	46	101110	(4, 4, 4, 4, 0, 1, 0, 3)	11520
20	47	101111	(4, 0, 4, 0, 0, 1, 0, 3)	36288
21	49	110001	(4, 0, 0, 0, 0, 0, 0, 0)	3648
22	51	110011	(4, 0, 0, 2, 0, 0, 0, 2)	11520
23	53	110101	(4, 0, 0, 1, 0, 0, 0, 1)	11520
24	54	110110	(4, 4, 4, 7, 0, 0, 0, 3)	576

25	55	110111	(4, 0, 0, 3, 0, 0, 0, 3)	36288
26	59	111011	(4, 0, 0, 0, 0, 0, 0, 2)	229824
27	61	111101	(4, 0, 0, 0, 0, 0, 0, 1)	229824
28	62	111110	(4, 4, 4, 4, 0, 0, 0, 3)	229824
29	63	111111	(4, 0, 0, 0, 0, 0, 0, 3)	15894144

These computational results agree with theoretical results given in [D1,S3]. The remaining classes for r = 3 are all empty. This can be seen also by examining the intersection properties of C-maximal sets (see section IV.4). Below we describe and list the empty classes from each intersection property.

From Lemma IV.4.3 it follows that classes such that $(e_4, e_5, e_6, e_j) = (0, 0, 0, 1)$, $1 \leq j \leq 3$, are empty. There are 7 such classes: **8**=001000, **16**=010000, **24**=011000, **32**=100000, **40**=101000, **48**=110000, **56**=111000.

From Lemma IV.4.4 it follows that classes such that (e_1, e_2, e_6) or (e_1, e_3, e_5) or $(e_2, e_3, e_4) = (0, 0, 1)$ are empty.There are 19 such classes: **1**=000001, **2**=000010, **3**=000011, **4**=000100, **5**=000101, **6**=000110, **7**=000111, **9**=001001, **11**=001011, **13**=001101, **15**=001111, **18**=010010, **19**=010011, **22**=010110, **23**=010111, **36**=100100, **37**=100101, **38**=100110, **39**=100111.

From Lemma IV.4.5 it follows that classes such that $(e_1, e_2, e_3, e_j) = (0, 0, 0, 1)$, $4 \leq j \leq 6$, are empty. There are 7 such classes: **1**=000001, **2**=000010, **3**=000011, **4**=000100, **5**=000101, **6**=000110, **7**=000111.

From Lemma IV.4.6 it follows that classes such that (e_4, e_5, e_3) or (e_4, e_6, e_2) or $(e_5, e_6, e_1) = (0, 0, 1)$ are empty. There are 19 such classes: **8**=001000, **9**=001001, **16**=010000, **18**=010010, **24**=011000, **25**=011001, **26**=011010, **32**=100000, **36**=100100, **40**=101000, **41**=101001, **44**=101100, **48**=110000, **50**=110010, **52**=110100, **56**=111000, **57**=111001, **58**=111010, **60**=111100.

Theorem V.2.1: [D1,S3] There exist 29 (4) classes of functions of 3-valued (2-valued, respectively) set logic under compositions with C functions (Boolean functions with constants).

Remark: a given class of (set logic) functions may not contain p-place functions $f(x_1, ..., x_p)$ but have q-place functions $f(x_1, ..., x_q)$ for $p \neq q$.

We also have used the same algorithms to classify and enumerate one-place set logic functions under compositions with B functions (Boolean functions without constants) (see section VI.4). Appendix 1 of the thesis contains all the Pascal codes for the classification and enumeration of set logic functions under compositions with C or B functions.

V.3. (S-)completeness and enumeration of (S-)bases

Once we have all the non empty classes of (set logic) functions, we can discuss the (S-)completeness properties in P_k in terms of these classes instead of individual functions; if a set is (S-)complete, then by replacing a function in the set by any function in the corresponding equivalence class yields another (S-)complete set. If $f \in F \subseteq P_k$ and $f \equiv g$, then clearly F is (S-)complete in P_k if and only if $(F \setminus \{f\}) \cup \{g\}$ is (S-)complete in P_k. In other words, it suffices to study the (S-)completeness in P_k up to the equivalence $\equiv$. It is easy to see that $F \subseteq P_k$ is (S-)complete in P_k if and only if $s = \sum_{f \in F} \varphi(f)$ (bitwise or operation) has all coordinates equal to 1 (i.e. in decimal, $s = 2^m - 1$, m = number of (S-)maximal sets in P_k). Once we know all the characteristic vectors, we can find all (S-)complete sets in P_k and all (S-)bases by a direct combinatorial check. If we associate $A = \{i : s_i = 1\}$ to $s_1...s_m \in \{0, 1\}^m$ and if $A_1, ..., A_t$ are the subsets of $\{1, ..., m\}$ corresponding to the characteristic vectors, the (S-)completeness problem is reduced to the listing of subsets of $\{A_1, ..., A_t\}$ covering $\{1, ..., m\}$ and the basis problem to the listing of such coverings which are irredundant (no proper subset covers $\{1, ..., m\}$).

V.3.a. Generating all t-subsets of {1, ..., n} in lexicographic order

We use the same algorithm described in [S5] to enumerate the C-bases. It is based on the lexicographic generation of all t-subsets (subsets containing t elements) of the set $\{1, ..., n\}$ and using bitwise redundancy check. Each subset is represented as a sequence $c_1...c_t$, $1 \le t \le n$, where $1 \le c_1 < ... < c_t \le n$ and where c_j is the ordinal number of a class of (set logic) functions for $j = 1, ..., t$.

Recall the definition of lexicographic order of subsets. For two subsets $a = (a_1, ..., a_p)$ and $b = (b_1, ..., b_q)$, $a < b$ is satisfied if and only if there exists i ($1 \le i \le q$) such that $a_j = b_j$ for $1 \le j < i$ and either $a_i < b_i$ or $p = i - 1$. For example, for $n = 5$, the lexicographic enumeration of t-subsets goes in the following manner:

1	12	123	1234	12345
			1235	
		124	1245	
		125		
	13	134	1345	
		135		
	14	145		
	15			

2	23	234	2345
		235	
	24	245	
	25		
3	34	345	
	35		
4	45		
5			

The algorithm is in *extend* phase when it goes from left to right staying in a row. It is in *reduce* phase when the algorithm shifts to the next row.

```
BEGIN
    read(n);
    t := 0;
    c_t := 0;
    REPEAT
        IF c_t < n THEN
            Extend
        ELSE
            Reduce;
        Print_out(c_1, ..., c_t);
    UNTIL c_t := n;
END.

Extend
BEGIN
    c_{t+1} := c_t + 1;
    t := t + 1;
END.

Reduce
BEGIN
    t := t - 1;
    c_t := c_t + 1;
END.
```

V.3.b. Enumeration of (classes of) (S-)bases

The lexicographic algorithm is applied in basis enumerations for a subset of P_k. We need the following definitions to describe the algorithm for enumeration of (S-)bases. Let m be the number of (S-)maximal sets in P_k. A set of (set logic) functions $F = \{f_1, ..., f_s\}$ is called *nonredundant (S-nonredundant* in set logic*)* in P_k, if for each i, $1 \le i \le s$, there exists a (S-)maximal set S_j in P_k, $1 \le j \le m$ which does not contain f_i while all the other functions f_t ($t = 1, ..., s, t \ne i$) are elements of S_j. In other words, F is (S-)nonredundant if and only if for each $g \in$

F the sum $s = \sum_{f \in F-\{g\}} \varphi(f)$ (bitwise or operation) has at least one 0 coordinate ((S-)nonredundancy condition). We call (S-)nonredundant incomplete sets simply *addable (S-addable* in set logic*)*. The rank of a (S-)addable set is the number of its elements. A class of (S-)bases also called an *aggregate* is the set of all (S-)bases having the same set of characteristic vectors. For a (S-)base, its class of (S-)bases is the set of classes of functions for functions belonging to the (S-)base.

Conditions of (S-)completeness and (S-)nonredundancy of a set F can be conveniently expressed by using characteristic vectors of set logic functions belonging to F. From the definitions of (S-)completeness, (S-)nonredundancy and (S-)base, it follows that there are two conditions for $F \subseteq P_k$ to be a (S-)base: (S-)completeness and (S-)nonredundancy. A (S-)base corresponds to a minimal cover of 1...1 (unit vector), and a (S-)nonredundant set corresponds to a minimal cover of some non-unit vector in which some 0's may occur (we except null vector). Using the bitwise or operation $\vee$ for characteristic vectors ($a_1...a_m \vee b_1...b_m = a_1 \vee b_1...a_m \vee b_m$), criteria for the (S-)completeness and (S-)nonredundancy of a set $c_1, ..., c_t$ of characteristic vectors are respectively the following [S5]:

$c_1 \vee ... \vee c_t$	$=$	1...1		(S-)completeness
$c_1 \vee ... \vee c_{j-1} \vee c_{j+1} \vee ... \vee c_t$	$\neq$	$c_1 \vee ... \vee c_t$,	for $1 \le j \le t$	(S-)nonredundancy

If we have a complete list of characteristic vectors for non empty classes of (set logic) functions of a set, we can enumerate all its aggregates. As an example, assume a set M contains 4 (S-)maximal sets M_1, M_2, M_3, M_4 and 6 classes of (set logic) functions:

1. 0011 2. 0100 3. 1000 4. 0010 5. 0001 6. 0000

For instance, class 1 is the set $M_1M_2\overline{M}_3\overline{M}_4$, where $\overline{M}_j = M \setminus M_j$ (complement). M has exactly two aggregates {1, 2, 3} and {2, 3, 4, 5}. Bitwise or operation for the set {1, 2, 3} results 1111 ((S-)completeness). Bitwise or operation for the set {1, 2} results 0111, for the set {1, 3} results 1011 and for the set {2, 3} results 1100 ((S-)nonredundancy). The set {1, 3, 4} is S-redundant, because bitwise or operation for the sets {1, 3, 4} and {1, 3} are equal (to 1011).

The lexicographic algorithm enumerates classes of (S-)bases ((S-)addables) sets for every rank at the same time. Moreover the maximal ranks of (S-)bases ((S-)addables) sets are automatically given as a result.

Let m and n denote the numbers of (S-)maximal sets and classes of (set logic) functions respectively. Suppose we are enumerating taken t elements out of n characteristic vectors stored in an array v as v(1), ..., v(n). The selected indexes are to be stored in an array c as c_1, ..., c_t, $1 \le c_i \le n$ for each i, $1 \le i \le t$. Suppose we are examining taken t-subset $v(c_1)$, ..., $v(c_t)$, where $1 \le c_1 < ... < c_t \le n$. There are three possible cases after the examination: the t-subset is either (S-)redundant or (S-)addable or a (S-)base set. The enumeration of subsets in lexicographic order can be controlled in the following manner. If a t-subset is either (S-)redundant or (S-)base then it is unnecessary to *extend* it to a t+1-subset, since adding a new vector to them will result in *(S-)redundancy*; in the former case the t-subset is already *(S-)redundant* and in the latter it is already *(S-)complete*. Hence in these cases we can bypass the lexicographic enumeration of subsets to an appropriate point; we say that the algorithm is in *cut* phase. The next subset is $c_1, c_2, ..., c_t - 1, c_t + 1$ if $c_t \ne n$; otherwise it is the next subset in lexicographic order and the bypass effects nothing. Thus the only remaining (S-)addable cases can be extended. The cut technique reduces the number of examined subsets by avoiding to generate (S-)redundant subsets; the algorithm goes from some subset to some subset in a lower row skipping several subsets.

As an example we consider the same set M as before. The class 6 is omitted. In this case m = 4 and n = 5. The notions extend, reduce, cut, redundant, base, addable, we denote simply by e, r, c, n, b, a respectively.

1ae	12ae	123bc		(123 is an aggregate of rank 3)
		124nc		
		125ncr		
	13ae	134nc		
		135ncr		
	14nc			
	15nc			
2ae	23ae	234ae	2345bcr	(2345 is an aggregate of rank 4)
		235ar		
	24ae	245ar		
	25ar			
3ae	34ae	345ar		
	35ar			
4ae	45ar			
5a				

[S5] writes the algorithm for basis enumeration as follows. Let b_t be the number of aggregates of rank t.

```
BEGIN
    read n, m, v(i) for i = 1, ..., n;
    r := 1;
    c_1 := 1;
    b_t := 0 for t = 1, ..., m;
    REPEAT
        IF v(c_1), ..., v(c_t) is addable THEN
            IF c_t < n THEN
                Extend
            ELSE
                Reduce
        ELSE
            IF v(c_1), ..., v(c_t) is a base THEN
                b_t := b_t + 1;
                Cut;
    UNTIL c_1 = n;
    Print_out(b_i), 1 ≤ i ≤ m;
END.

Cut
BEGIN
    IF c_t < n THEN
        c_t := c_t + 1
    ELSE
        Reduce;
END.
```

In the algorithm Reduce and Extend are defined as before. Note that the last set n is not checked. This can be done by putting *UNTIL t = 0* instead of *UNTIL* $c_1 = n$.

If the number of (S-)maximal sets m is less than the number of bits in a register of a given computer, the characteristic vectors (which are binary numbers) can be converted into decimal numbers. In the examination of a t-subset (i.e. a set of t vectors) for (S-)base or (S-)addable set, we can treat these vectors as integer numbers because OR operation between integer numbers is defined as a machine instruction OR between corresponding components (bits) of their binary notations. Therefore, the checks for (S-)redundancy, (S-)addable or (S-)base become easier and faster. Otherwise bitwise or operation can be realized with characteristic vectors as an array of m elements.

```
Check för a Base or Addable set
BEGIN
    IF v(c_1) ∨ ... ∨ v(c_t) = 2^m - 1 THEN
            Is Base := True                    {= 11...11, all m coordinates are 1}
    ELSE
            Is Addable := True;                {= ...0..., there is at least one coordinate 0}
END.
```

V.3.c. Redundancy checks

[S5] describes also a technique, called *bitwise redundancy checks*, to reduce the computation in (S-)redundancy checks.

For simplicity we will write v_i for $v(c_i)$. Suppose we are checking (S-)redundancy of v_1, ..., v_t. For every (S-)redundancy check we know that v_1, ..., v_{t-1} are included in the subset which we examined before (only v_t is a newly added vector). Thus we can assume that we already have $T_s = v_1 \vee ... \vee v_s$ for $1 \leq s \leq t - 1$ in an array T (for convenience we add T_0 and assume $T_0 = 0$). The (S-)redundancy condition for the t-subset can be formulated in the following way (a variable B is used to reduce the number of bitwise or operations).

```
Redundancy                                     {for t ≥ 2}
BEGIN
    B := 0;
    T_0 := 0;
    T_t := T_t-1 ∨ v_t;
    IF T_t-1 = T_t THEN
            Redundancy := true
    ELSE
            FOR s := t - 1, ..., 1 DO          {s ≥ 2}
                    B := B ∨ v_s+1;
                    IF T_s-1 ∨ B = T_t THEN
                            Redundancy := True;
END.
```

For $t = 1$, the vector v_1 is addable if it is neither null nor unit vector. A unit vector is a base and a null vector is considered redundant.

When the number of classes is large the characteristic vectors must be sorted according to their number of units in non increasing order, in order to minimize the running time. Finally our basis enumeration algorithm becomes

```
BEGIN
    read n, m, v(i) for i = 1, ..., n;
    r := 1;
    c_1 := 1;
    b_t := 0 for t = 1, ..., m;
    sort the vectors according to their number of 1's in non increasing order;
    REPEAT
            IF v(c_1), ..., v(c_t) is Redundant THEN
                    Cut
            ELSE
                    IF v(c_1), ..., v(c_t) is Addable THEN
                            IF c_t < n THEN
                                    Extend
                            ELSE
                                    Reduce
                    ELSE
                            IF v(c_1), ..., v(c_t) is a Base THEN
                                    b_t := b_t + 1;
                                    Cut;
    UNTIL t = 0;
    Print_out(b_t), 1 ≤ t ≤ m;
END.
```

V.4. Enumeration of C-bases and C-Sheffer functions in P_4 and P_8

P_4 has exactly two C-maximal sets: E_1 and E_2 (see section IV.3). There are also four classes of set logic functions (1.00, 2.01, 3.10, 4.11). It is easy to see that the set {2, 3} of classes forms one aggregate (class of C-bases) of rank two while the set {4} contains C-Sheffer functions for P_4. Therefore the maximum rank of any C-base of P_4 is 2. [S3] gives the following results.

Theorem V.4.1:

(i) The number of n-place C-Sheffer functions in P_4 is

$$2^{2^{2n+1}} - 2^{2^{2n}+2^n+1} + 2^{2^n+1}.$$

(ii) The number of C-bases of rank two containing n-place functions in P_4 is

$$2^{2^{2n+1}+2^n+1} - 2^{2^{2n}+2^n+1+2^n+1} + 2^{2^n+2}.\blacksquare$$

In general, the number of (S-)bases of P_k consisting of n-place (set logic) functions in an aggregate can be calculated as product of the numbers of n-place functions in the classes of

functions determined by the characteristic vectors. Summing these numbers for all aggregates for a rank we obtain corresponding data for the (S-)bases of the rank and finally the number of all (S-)bases consisting of n-place set logic functions. For instance, classes 2, 3 and 4 contain 48, 48 and 144 one-place functions of P_4 respectively (see section V.2). The number of C-bases of rank two in the aggregate {2, 3} consisting of one-place functions is $|\{2, 3\}| = |\{2\}| \times |\{3\}| = 48 \times 48 = 2304$ and the total number of C-bases in P_4 consisting of one-place functions is 2304 + 144 = 2448.

There are six C-maximal sets and 29 classes of set logic functions in P_8 (see section V.2). Using the algorithm for enumeration of C-bases in P_8 we obtain the following results (the numbers here are the ordinal numbers of the 29 classes as they appear in section V.2; the classes are sorted according to their number of units).

1 aggregate of rank 1 : (29)

120 aggregates of rank 2 : (28 27) , (28 26) , (28 25) , (28 20) , (28 12) , (28 23) , (28 22) , (28 18) , (28 17) , (28 10) , (28 8) , (28 21) , (28 15) , (28 7) , (28 13) , (28 5) , (27 26) , (27 25) , (27 20) , (27 12) , (27 24) , (27 22) , (27 19) , (27 17) , (27 11) , (27 8) , (27 16) , (27 15) , (27 4) , (27 14) , (27 2) , (26 25) , (26 20) , (26 12) , (26 24) , (26 23) , (26 19) , (26 18) , (26 11) , (26 10) , (26 9) , (26 7) , (26 4) , (26 6) , (26 3) , (25 20) , (25 12) , (25 19) , (25 18) , (25 17) , (25 11) , (25 10) , (25 8) , (25 16) , (25 9) , (25 4) , (25 3) , (25 2) , (20 12) , (20 24) , (20 23) , (20 22) , (20 11) , (20 10) , (20 8) , (20 21) , (20 9) , (20 7) , (20 6) , (20 5) , (12 24) , (12 23) , (12 22) , (12 19) , (12 18) , (12 17) , (12 21) , (12 16) , (12 15) , (12 14) , (12 13) , (24 18) , (24 17) , (24 10) , (24 8) , (23 19) , (23 17) , (23 11) , (23 8) , (23 16) , (23 4) , (23 2) , (22 19) , (22 18) , (22 11) , (22 10) , (22 9) , (22 4) , (22 3) , (19 10) , (19 8) , (19 21) , (19 7) , (19 5) , (18 11) , (18 8) , (17 11) , (17 10) , (17 9) , (17 7) , (17 6) , (11 21) , (11 15) , (11 13) , (10 16) , (10 15) , (10 14) , (21 4) , (16 7) , (15 9)

175 aggregates of rank 3 : (24 23 11) , (24 23 16) , (24 23 9) , (24 23 3) , (24 22 11) , (24 22 16) , (24 22 4) , (24 22 2) , (24 19 21) , (24 19 15) , (24 19 13) , (24 11 21) , (24 11 7) , (24 11 5) , (24 21 16) , (24 21 9) , (24 21 3) , (24 21 2) , (24 16 15) , (24 16 13) , (24 16 5) , (24 15 4) , (24 15 3) , (24 15 2) , (24 9 7) , (24 9 13) , (24 9 5) , (24 7 4) , (24 7 3) , (24 7 2) , (24 4 13) , (24 4 5) , (24 13 3) , (24 13 2) , (24 5 3) , (24 5 2) , (23 22 18) , (23 22 10) , (23 22 9) , (23 22 3) , (23 18 15) , (23 18 14) , (23 10 15) , (23 10 14) , (23 15 3) , (23 9 14) , (23 14 3) , (22 17 7) , (22 17 6) , (22 8 7) , (22 8 6) , (22 16 7) , (22 16 6) , (22 7 2) , (22 6 2) , (19 18 9) , (19 18 6) , (19 17 9) , (19 17 6) , (19 11 15) , (19 11 13) , (19 15 6) , (19 9 13) , (19 13 6) , (18 17 10) , (18 17 21) , (18 17 7) , (18 17 5) , (18 10 16) , (18 10 4) , (18 10 2) , (18 21 16) , (18 21 15) , (18 21 14) , (18 21 2) , (18 16 9) , (18 16 6) , (18 16 5) , (18 15 7) , (18 15 6) , (18 15 5) , (18 9 4) , (18 9 14) , (18 9 2) , (18 7 4) , (18 7 14) , (18 7 2) , (18 4 6) , (18 4 5) , (18 14 6) , (18 14 5) , (18 6 2) , (18 5 2) , (17 8 4) , (17 8 3) , (17 21 4) , (17 21 3) , (17 4 5) , (17 5 3) , (11 10 16) , (11 10 14) , (11 8 16) , (11 8 14) , (11 16 7) , (11 16 5) , (11 7 14) , (11 14 5) , (10 8 21) , (10 8 13) , (10 21 4) , (10 21 2) , (10 4 13) , (10 13 2) , (8 21 9) , (8 21 7) , (8 21 6) , (8 21 3) , (8 16 9) , (8 16 4) , (8 16 6) , (8 16 3) , (8 15 7) , (8 15 4) , (8 15 6) , (8 15 3) , (8 9 14) , (8 9 13) , (8 7 14) , (8 7 13) , (8 4 14) , (8 4 13) , (8 14 6) , (8 14 3) , (8 13 6) , (8 13 3) , (21 16 9) , (21 16 6) , (21 16 3) , (21 15 3) , (21 9 14) , (21 9 2) , (21 7 2) , (21 14 3) , (21 6 2) , (21 3 2) , (16 15 6) , (16 9 13) , (16 9 5) , (16 4 5) , (16 13 6) , (16 6 5) , (16 5 3) , (15 7 4) , (15 7 3) , (15 7 2) , (15 4 6) , (15 4 5) , (15 6 3) , (15 6 2) , (15 5 3) , (9 7 14) , (9 4 13) , (9 14 13) , (9 14 5) , (9 13 2) , (7 4 14) , (7 4 13) , (7 14 3) , (7 14 2) , (7 13 2) , (4 14 5) , (4 13 6) , (4 13 5) , (14 5 3) , (13 6 2)

7 aggregates of rank 4 : (21 15 7 2) , (21 15 6 2) , (21 14 6 2) , (16 15 4 5) , (14 13 6 3) , (14 6 5 2) , (13 5 3 2)

0 aggregates of rank 5 and 6.

Thus the rank of any C-base of P_8 is between 1 and 4 ([D1] gives a theoretical proof). The following table shows the number of C-bases of P_8 containing one-place functions for a given rank.

Ranks	1	2	3	4	Σ
Number of C-bases	15894144	2020225024	459800576	889192448	$\approx 3.385112 \cdot 10^9$

The computer found 15894144 as the number of one-place C-Sheffer functions for P_8. Suppose we have m (S-)maximal sets S_1, ..., S_m, then a n-place function f is (S-)Sheffer for P_k if and only if $f \notin S_i$ for $1 \le i \le m$ (or $\{f\} \not\subset S_i$, (S-)completeness criteria). That is $f \in P_k(n) \setminus (S_1(n) \cup ... \cup S_m(n))$. The number of n-place (S-)Sheffer functions is $|P_k(n) \setminus (S_1(n) \cup ... \cup S_m(n))|$. If m is small ($m \le 7$) and the cardinality of the S_i's is known, the number of (S-)Sheffer functions for P_k can be usually calculated using the principle of inclusion/exclusion and the known knowledges on the intersection properties of these (S-)maximal sets. For large m (say $m \ge 8$) it may be hard to use the principle of inclusion/exclusion because of the large number of intersections to consider. In that case, it may be sufficient to estimate the number of (S-)Sheffer functions, i.e. find a lower bound and an upper bound on that number.

Theorem V.4.2: [D1] The number of n-place C-Sheffer functions for P_8 is

$$8^{8^n} - 3 \cdot 2^{8^n} 4^{4^n} - 3 \cdot 4^{8^n} 2^{2^n} + 6 \cdot 2^{2^n} 2^{4^n} 2^{8^n} + 3 \cdot 4^{4^n} 2^{2^n} - 6 \cdot 2^{4^n} 2^{2^n + 1} + 2 \cdot 8^{2^n}. \blacksquare$$

It is easy to see that the ratio of n-place C-Sheffer functions over all n-place set logic functions, respectively in P_4 and P_8, approaches 1 as n grows. Therefore almost all set logic functions in P_4 and P_8 are C-Sheffer functions.

In the following chapter we classify the functions with respect to B-maximal sets and enumerate the B-bases and B-Sheffer functions in two-valued and three-valued set logic. Appendix 2 of the thesis contains all the Pascal codes for the classification and enumeration of (S-)bases.

CHAPTER VI
(MAIN CONTRIBUTION)
CLASSIFICATION OF FUNCTIONS AND ENUMERATION OF BASES OF SET LOGIC UNDER COMPOSITIONS WITH B FUNCTIONS

VI.1. Introduction

This chapter discusses some classification and enumeration problems in r-valued set logic, which is the logic of functions mapping n-tuples of subsets into subsets over r values. Boolean functions are convenient choice as building blocks in the design of set logic. B-maximal sets are maximal sets containing all B functions. We give the number of n-place functions in each B-maximal set and find some properties of intersection of B-maximal sets in r-valued set logic. These properties are used to classify all two-valued and three-valued set logic functions according to the B-maximal sets they belong to. We prove that there are 8 and 200 such classes of functions respectively in two-valued and three-valued set logic. For each class of functions we give a one-place example function and its total number of one-place set logic functions. Finally, we study the B-Sheffer functions, i.e. functions which are complete under compositions with B functions. We find the number of n-place B-Sheffer functions of two-valued set logic and give a lower bound and an upper bound on the number of n-place B-Sheffer functions of three-valued set logic. Also we enumerate all the classes of B-bases of two valued and three-valued set logic.

VI.2. B-completeness in set logic

In [A3,A4,A5,A6,A7] the question of constructing all set logic functions using Boolean functions is studied. Since the set is incomplete, some functions are added to the set of Boolean functions to form a complete set. In these considerations Boolean functions are considered *cheap* elements in the design of set logic functions. Following these investigations, it is interesting to characterize all sets of functions which become complete when all Boolean functions are added to them. The question has been studied in literature for the case of [C] functions and was defined as the problem of *weak* completeness in [S3] and Boolean completeness in [S4] (with the same meaning). Recall the S-completeness criterion.

Let S be a given set of *cheap* functions. A maximal set F in P_k is said to be S-maximal in P_k if $S \subseteq F$. A subset F of $P_k \setminus [S]$ is said to be S-complete in P_k if the set $S \cup F$ is complete in P_k. A S-complete set F in P_k is called a S-base of P_k if no proper subset of F is S-complete in P_k.

The rank of a S-base is the number of its elements. A function $f \in P_k$ is S-Sheffer (or Sheffer with S functions) for P_k if $\{f\} \cup S$ is complete in P_k, or equivalently, if $\{f\}$ is a S-base (of rank 1) of P_k. For example, bio-output [A7], conversion [A2] and literal [A5] functions are respectively $\{\cup, \bar{}, \backslash\}$-Sheffer, $\{\cup, \bar{}\}$-Sheffer and $\{\cup, \cap\}$-Sheffer for P_k with constants.

Theorem VI.2.1: [D2] (S-completeness theorem in a general form). A subset F of functions in P_k is S-complete in P_k if and only if F is contained in no S-maximal set in P_k.

For $S = B = \{\cup, \cap, \bar{}\}$, [D2] proved that there are Bl(r) + k - 3 B-maximal sets in set logic where Bl(r) is a Bell number, i.e. the number of set partitions of the set R. Among the Bl(r) + k - 3 B-maximal sets, k - 2 are defined by equivalence relations on L_k and Bl(r) - 1 by unary central relations on L_k.

For convenience, we denote the equivalence relations by ε_i, $1 \le i \le k - 2$, and corresponding B-maximal sets by $E_i = Pol\varepsilon_i$. The relation ε_i is defined as follows. One equivalence class of ε_i consists of all numbers j such that $x \subseteq j \subseteq i \cup x$, where $\subseteq$ is the set inclusion ($a \subseteq b$ iff $a_m \le b_m$ for $0 \le m \le r - 1$). The element x is any such that $i \cap x = 0$. Suppose that i has t 0's and r - t 1's. Then x in the last definition must have 0 at any position where i has 1 (for example, if $i = 3 = 011$ then $x = 0 = 000$ or $x = 4 = 100$) and thus there are 2^t equivalence classes of ε_i each containing 2^{r-t} elements. For example, for r = 3 the six equivalence relations which are preserved by all [C] and [B] functions are the following:

$$\varepsilon_1 = \{\{0, 1\}, \{2, 3\}, \{4, 5\}, \{6, 7\}\}$$
$$\varepsilon_2 = \{\{0, 2\}, \{1, 3\}, \{4, 6\}, \{5, 7\}\}$$
$$\varepsilon_3 = \{\{0, 1, 2, 3\}, \{4, 5, 6, 7\}\}$$
$$\varepsilon_4 = \{\{0, 4\}, \{1, 5\}, \{2, 6\}, \{3, 7\}\}$$
$$\varepsilon_5 = \{\{0, 1, 4, 5\}, \{2, 3, 6, 7\}\}$$
$$\varepsilon_6 = \{\{0, 2, 4, 6\}, \{1, 3, 5, 7\}\}.$$

For r = 2 we have 2 equivalence relations preserved by [C] and [B] functions: $\varepsilon_1 = \{\{0, 1\}, \{2, 3\}\}$, $\varepsilon_2 = \{\{0, 2\}, \{1, 3\}\}$.

If the elements of L_k are drawn as edges of a r-dimensional hypercube then the equivalence relations ε_i are all partitions of the hypercube into equal size subcubes. The subdivision is uniquely determined by the subcube containing 0, and the subcube containing 0 is uniquely determined by its largest element (between 1 and k - 2).

We denote the unary central relations by γ_j, $0 \le j \le Bl(r) - 2$, and the corresponding B-maximal sets by $C_j = Pol\gamma_j$. The relation γ_j is any subset of L_k having the following properties: if $a \in \gamma_j$ then $\bar{a} \in \gamma_j$; if $a, b \in \gamma_j$ then $a \cap b \in \gamma_j$. Each element of γ_j is a subset of basic r-values $R = \{e_0, e_1, ..., e_{r-1}\}$ where e_i is 0 or 1. Thus γ_j corresponds to a subalgebra of Boolean algebra on R. Every set partition of R defines one such subalgebra as follows. If e_a and e_b are in the same class of given set partition then they either both are 0 or both are 1 in each element of γ_j. γ_j contains all such elements (therefore 0 and k - 1 in all cases). Every set partition (except the trivial one when all elements are in the same set) defines one subset γ_j corresponding to a B-maximal set. The number of elements in γ_j is 2^m where m is the number of equivalence classes in the set partition. For convenience, we let $\gamma_0 = \{0, k - 1\}$ denote the first unary central relation and also let $C_0 = Pol\gamma_0$ be the corresponding B-maximal set. For example, for r = 3 there exist four unary central relations preserved by all [B] functions:

$$\gamma_0 = \{0, 7\} \qquad \gamma_1 = \{0, 1, 6, 7\}$$
$$\gamma_2 = \{0, 2, 5, 7\} \qquad \gamma_3 = \{0, 3, 4, 7\}.$$

For r = 2 we have only one unary central relation preserved by [B] functions: $\gamma_0 = \{0, 3\}$.

In the case of γ_3 we may use the index 4 (i.e. γ_4) for convenience instead of the index 3 (see the text following Lemma VI.3.1).

VI.3. Enumeration of functions and intersection properties of B-maximal sets in set logic.

In this section we first show some properties which involve B-maximal sets corresponding to unary central relations on L_k. Properties involving only B-maximal sets defined from equivalence relations will follow. In the sequel, we let B_s denote any B-maximal set for s = 1, 2, ..., Bl(r) + k - 3. Whenever it is possible to avoid confusion we shall call unary central relations only central relations.

Theorem VI.3.1: If $C = Pol\gamma$, where γ is a central relation corresponding to B-maximal set then

$$|C(n)| = |\gamma|^{|\gamma|^n} \cdot (2^r)^{(2^r)^n - |\gamma|^n}.$$

Proof: Let f be a n-place function of P_k. Partition the set L_k^n into two subsets S_1 and S_2, where S_1 is the set containing all points $x = (x_1, x_2, ..., x_n) \in L_k^n$ such that $x_i \in \gamma$, for i = 1, 2, ..., n,

and, S_2 is the complement of S_1 in L_k^n. If $f \in C$ then the values $f(x)$ in the points of S_1 are all from γ. The value in each point of S_1 can be chosen in $|\gamma|$ ways. And as the number of points in S_1 is $|\gamma|^n$, therefore we have $|\gamma|^{|\gamma|^n}$ ways to fix the values in the points of S_1. The values in the points of S_2 are all taken from L_k, that is the value in each point of S_2 can be chosen in 2^r ways. The number of points in S_2 is $(2^r)^n - |\gamma|^n$, and then there are $(2^r)^{(2^r)^n - |\gamma|^n}$ ways to fix the values in the points of S_2. It follows that the total number of functions which preserves the central relation γ is $|\gamma|^{|\gamma|^n} \cdot (2^r)^{(2^r)^n - |\gamma|^n}$. This completes the proof.■

Lemma VI.3.2: If the binary representation of i has t 0's and $t \geq 2$, then for $r \geq 3$

$$\text{Pol}\gamma_i \text{ is B-maximal, where } \gamma_i = \{j \mid 0 \subseteq j \subseteq i\} \cup \{j \mid \bar{i} \subseteq j \subseteq k - 1\}.$$

Proof: Recall that a unary central relation ρ corresponding to a B-maximal set is a proper subset of L_k having the following properties: if $a \in \rho$ then $\bar{a} \in \rho$; if $a, b \in \rho$ then $a \cap b \in \rho$. Let $eq_0 = \{j \mid 0 \subseteq j \subseteq i\}$ and $eq_{k-1} = \{j \mid \bar{i} \subseteq j \subseteq k - 1\}$. Therefore, if $x \in eq_0$ (eq_{k-1}) then $0 \subseteq x \subseteq i$ ($\bar{i} \subseteq x \subseteq k - 1$) and thus $\bar{i} \subseteq \bar{x} \subseteq k - 1$ ($0 \subseteq \bar{x} \subseteq i$), i.e. $\bar{x} \in eq_{k-1}$ (eq_0, resp.). Also, if $x, y \in eq_0$ (eq_{k-1}) then $x \cap y \in eq_0$ (eq_{k-1}, resp.); if $x \in eq_0$ (eq_{k-1}) and $y \in eq_{k-1}$ (eq_0) then $x \cap y \in eq_0$. As $t \geq 2$, then $\gamma_i = eq_0 \cup eq_{k-1}$ is a proper subset of L_k. It follows that $\text{Pol}\gamma_i$ is B-maximal.■

Since the equivalence classes eq_0 and eq_{k-1} are always in the relation ε_i then for convenience, we say that γ_i is defined from ε_i. In general for $r \geq 3$, $k - 2 - r$ central relations corresponding to B-maximal sets can be found in such a way. For example in P_{16} we have 14 central relations and 10 (= 16 - 2 - 4) among them are defined from 10 equivalence relations corresponding to B-maximal sets. (Remark: it is easy to see that γ_i contains 2^{r-t+1} elements).

Lemma VI.3.3: If the binary representation of i has t 0's and $t \geq 2$, then for $r \geq 3$

$$E_i \cap C_0 \subset C_i = \text{Pol}\gamma_i, \text{ where } \gamma_i = \{j \mid 0 \subseteq j \subseteq i\} \cup \{j \mid \bar{i} \subseteq j \subseteq k - 1\}.$$

Proof: Let f be a n-place function of P_k such that $f \in E_i \cap C_0$. We want to prove that $f \in C_i$. Partition the set L_k^n into n-dimensional blocks in such a way that two n-tuple points $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$, $x_j, y_j \in L_k$, belong to the same block iff for every $j = 1, 2, ..., n$, both x_j and y_j are in the same equivalence class of ε_i. The set γ_0^n contains 2^n points and intersects 2^n blocks of the partition, each in exactly one point we call an intersect-point. The relation ε_i contains the two equivalence classes $eq_0 = \{j \mid 0 \subseteq j \subseteq i\}$ and $eq_{k-1} = \{j \mid \bar{i} \subseteq$

$j \subseteq k - 1\}$ which construct all the 2^n intersect-blocks. Therefore, the points in the intersect-blocks are all from γ_i^n. As $f \in C_0$, the value $f(x)$ in an intersect-point x is from γ_0, i.e. $f(x) = 0$ or $f(x) = k - 1$. As $f \in E_i$, the value $f(y)$ in any other point y of the intersect-block is from the same equivalence class of ε_i containing $f(x)$. Thus, either both $f(x), f(y) \in eq_0$ or $f(x), f(y) \in eq_{k-1}$, i.e. $f(x), f(y) \in \gamma_i$. Therefore, for any n-tuple point $z \in L_k^n$, we have $f(z) \in \gamma_i$ whenever $z \in \gamma_i^n$. It follows that $f \in C_i$ and then $E_i \cap C_0 \subset C_i$.■

The c-th coordinate i_c of i is called fixed (free) if $i_c = 0$ ($ic = 1$). Given an equivalence class eq_i, all elements in eq_i have the same c-th coordinate whenever c is a fixed coordinate (i.e. if x and y are in the same equivalence class of ε_i and $i_c = 0$ then $x_c = y_c$).

Lemma VI.3.4: If $r = 3$ and $j \subset i$ then

$$E_i \cap E_{\bar{i}} \cap C_j \subset E_j, \text{ where } C_j = Pol\gamma_j \text{ and } \gamma_j = \{h \mid 0 \subseteq h \subseteq j\} \cup \{h \mid \bar{j} \subseteq h \subseteq k - 1\}.$$

Proof: Because of symmetry we can specify $j = 1$ and $i = 5$, that is $E_5 \cap E_2 \cap C_1 \subset E_1$ for example. Let f be a n-place function of P_k such that $f \in E_5 \cap E_2 \cap C_1$. Given two points $x = (x_1, ..., x_n)$ and $y = (y_1, ..., y_n)$, suppose $(x_c, y_c) \in \varepsilon_1$, $1 \le c \le n$, but $(f(x), f(y)) \notin \varepsilon_1$. As $(f(x), f(y)) \in \varepsilon_5$ and $(f(x), f(y)) \notin \varepsilon_1$ we have either $f(x) \in \{0, 1\}$, $f(y) \in \{4, 5\}$, or $f(x) \in \{2, 3\}$, $f(y) \in \{6, 7\}$, or $f(x) \in \{4, 5\}$, $f(y) \in \{0, 1\}$, or $f(x) \in \{6, 7\}$, $f(y) \in \{2, 3\}$. Thus either $f(x) \in \gamma_1$ and $f(y) \notin \gamma_1$ or vice versa. Assume $f(x) \in \gamma_1$, i.e. either $f(x) \in \{0, 1\}$, $f(y) \in \{4, 5\}$, or $f(x) \in \{6, 7\}$, $f(y) \in \{2, 3\}$. Define z and w such that $(x_c, z_c) \in \varepsilon_2$, $(y_c, w_c) \in \varepsilon_2$, and $z_c, w_c \in \gamma_1$. Clearly, it is easy to see that $(z_c, w_c) \in \varepsilon_5$, $1 \le c \le n$. As $f(x), f(z) \in \gamma_1$ and $(f(x), f(z)) \in \varepsilon_2$ it follows that $f(x) = f(z)$. Therefore, for $f(x) \in \{0, 1\}$ and $f(y) \in \{4, 5\}$ we have $f(z) \in \{0, 1\}$ and $f(w) \in \{6, 7\}$, and for $f(x) \in \{6, 7\}$, $f(y) \in \{2, 3\}$ we have $f(z) \in \{6, 7\}$ and $f(w) \in \{0, 1\}$. Thus we have $(z_c, w_c) \in \varepsilon_5$ and $(f(z), f(w)) \notin \varepsilon_5$, which is a contradiction since $f \in E_5$. Consequently $E_i \cap E_{\bar{i}} \cap C_j \subset E_j$.■

Lemma VI.3.5: Let t(i) and t(j) denote respectively the number of 0's in the binary representations of i and j. If $t(j) \ge 2$ and ($j \subset i$ and $t(i) = t(j) - 1$) or ($j \subset \bar{i}$ and $t(\bar{i}) = t(j) - 1$) then

$$E_i \cap E_{\bar{i}} \cap C_j \subset E_j, \text{ where } C_j = Pol\gamma_j \text{ and } \gamma_j = \{h \mid 0 \subseteq h \subseteq j\} \cup \{h \mid \bar{j} \subseteq h \subseteq k - 1\}.$$

Proof: This Lemma is a generalization of Lemma VI.3.4. However the proof is too long and quite complicated. We are searching to reduce it so that the final version will be added here.

Lemma VI.3.6: If Polγ and Polγ' are B-maximal sets from distinct central relations then Pol($\gamma \cap \gamma'$) is also B-maximal.

Proof: Recall that a unary central relation ρ corresponding to a B-maximal set is a proper subset of L_k having the following properties: if $a \in \rho$ then $\bar{a} \in \rho$; if $a, b \in \rho$ then $a \cap b \in \rho$. Now suppose that $a \in \gamma \cap \gamma'$. This implies that $a \in \gamma$ and $a \in \gamma'$. As γ and γ' are central relations corresponding to B-maximal sets, therefore $\bar{a} \in \gamma$ and $\bar{a} \in \gamma'$, and then $\bar{a} \in \gamma \cap \gamma'$. Next, if $a \in \gamma \cap \gamma'$ and $b \in \gamma \cap \gamma'$ then we have $a, b \in \gamma$ and $a, b \in \gamma'$. Again, as γ and γ' are central relations, therefore $a \cap b \in \gamma$ and $a \cap b \in \gamma'$, and then $a \cap b \in \gamma \cap \gamma'$. It follows that $\gamma \cap \gamma'$ is a central relation corresponding to a B-maximal set.▪

Lemma VI.3.7: Pol$\gamma \cap$ Pol$\gamma' \subset$ Pol($\gamma \cap \gamma'$).

Proof: Let f be a n-place function of P_k such that $f \in$ Pol$\gamma \cap$ Polγ'. Let $x = (x_1, x_2, ..., x_n) \in L_k^n$ such that $x_i \in \gamma \cap \gamma'$, $1 \le i \le n$. $x_i \in \gamma \cap \gamma'$ implies $x_i \in \gamma$ and $x_i \in \gamma'$. As $f \in$ Polγ (Polγ') then $x_i \in \gamma$ (γ') implies $f(x) \in \gamma$ (γ', resp.). It follows that $f(x) \in \gamma \cap \gamma'$ whenever $x_i \in \gamma \cap \gamma'$, and then $f \in$ Pol($\gamma \cap \gamma'$). Therefore Pol$\gamma \cap$ Pol$\gamma' \subset$ Pol($\gamma \cap \gamma'$).▪

Lemma VI.3.8: For r = 2 we have

$$|\cap B_s(n)| = |E_1(n) \cap E_2(n) \cap C_0(n)| = 2^{2^n}.$$

Proof: $\cap B_s(n)$ is the set of n-place functions preserving relations ε_1, ε_2 and γ_0. Let $f \in \cap B_s(n)$. Consider two distinct partitions of the set L_4^n. Partition π_i (i = 1, 2) is performed in such a way that two n-tuple points $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$, $x_m, y_m \in L_4$, belong to the same block iff for every m = 1, 2, ..., n, both x_m and y_m belong to the same equivalence class of ε_i. The set γ_0^n contains 2^n points and intersects all 2^n blocks of π_i each in exactly one point we call an intersect-point (because 0 and 3 are always in two distinct equivalence classes for each ε_i). It is easy to see that any two blocks from distinct partitions, π_1 and π_2, have exactly one common point (note that all intersections $\{0, 1\} \cap \{0, 2\} = \{0\}$, $\{0, 1\} \cap \{1, 3\} = \{1\}$, $\{2, 3\} \cap \{0, 2\} = \{2\}$, $\{2, 3\} \cap \{1, 3\} = \{3\}$ are single sets). So after choosing the values f(x) in the intersect-points from γ_0, the function f is uniquely

determined (i.e. the common point between two blocks b_{π_1} and b_{π_2}, of π_1 and π_2 respectively, must map to the common element between the equivalence class of ε_1 and the equivalence class of ε_2 containing the value of their respective intersect-point). It follows that the number of functions preserving both relations ε_1, ε_2 and γ_0 is 2^{2^n}. This completes the proof.▪

Lemma VI.3.9: For $r \geq 3$

$$|\cap B_s(n)| = |E_1(n) \cap ... \cap E_{k-2}(n) \cap C_0(n) \cap ... \cap C_{Bl(r)-2}(n)| = 2^{2^n}.$$

Proof: Let $f \in \cap B_s(n)$. $\cap B_s(n)$ is the set of n-place functions preserving all relations corresponding to B-maximal sets. From Lemmas VI.3.6 and VI.3.7 it follows that $\cap C_j \subset C_0$, $1 \leq j \leq Bl(r) - 2$. From [D1] we have $\cap E_a = BF = \cap E_{l(i)}$, where $l(i) = 2^r - 1 - 2^i$, $0 \leq i \leq r - 1$ and $1 \leq a \leq k - 2$. Therefore $\cap B_s(n) = BF_k(n) \cap \cap C_j(n) = \cap E_{l(i)}(n) \cap \cap C_j(n)$. Consider r distinct partitions of the set L_k^n. Partition $\pi_{l(i)}$ is performed in such a way that two points $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$, $x_m, y_m \in L_k$, belong to the same block iff for every $m = 1, 2, ..., n$, both x_m and y_m are in the same equivalence class of $\varepsilon_{l(i)}$. The set γ_0^n intersects each block of $\pi_{l(i)}$ in exactly one point we call an intersect-point (because 0 and $k - 1$ are always in two distinct equivalence classes for each $\varepsilon_{l(i)}$). Also γ_0^n intersects each set γ_j^n in 2^n points (because $\gamma_0 \subset \gamma_j$, $1 \leq j \leq Bl(r) - 2$). It is easy to see that any r blocks, taken one per partition, have exactly one common point (this is equivalent to say that any r mutually non-parallel (r-1)-dimensional faces of r-dimensional cube intersect in exactly one element; for example for $r = 3$, the classes $\{4, 5, 6, 7\}$ of ε_3, $\{0, 1, 4, 5\}$ of ε_5 and $\{1, 3, 5, 7\}$ of ε_6 have only the element 5 in common). As $f \in C_0$ the values in the intersect-points are all from γ_0. As the function $f \in \cap E_{l(i)}$ the common point p of r blocks $b_{\pi_{l(0)}}, ..., b_{\pi_{l(r-1)}}$, of the partitions $\pi_{l(0)}, ..., \pi_{l(r-1)}$, must map to the common element e of the equivalence class of $\varepsilon_{l(0)}, ..., \varepsilon_{l(r-1)}$ containing the value of the intersect-point of $b_{\pi_{l(0)}}, ..., b_{\pi_{l(r-1)}}$, respectively. Also it is easy to see that if $p \in \gamma_j^n$ then $e \in \gamma_j$ for $1 \leq j \leq Bl(r) - 2$ (otherwise $f \notin \cap B_s(n)$). So after choosing the values in the intersect-points from γ_0, the function f is uniquely determined. Since γ_0 contains 2 elements, it follows that the number of functions preserving all relations is 2^{2^n}.▪

$|\cap B_s(n)|$ does not depend on r (or k). Also, for $n = 1$ the four [B] functions preserving all relations are: the constant function $f(x) = 0$, the identity function $f(x) = x$, the complement function $f(x) = k - 1 - x$, and the constant function $f(x) = k - 1$.

Lemma VI.3.10: Let $B_k(n)$ be the set of all n-place [B] functions in P_k, then

$$|B_k(n)| \leq 2^{2^n}.$$

Proof: Clearly [B] $\subset B_s$ for s = 1, ..., Bl(r) + k - 3. Hence follows $B_k(n) \subset B_s(n)$ for every $n \geq 0$, and consequently $B_k(n) \subseteq \cap B_s(n)$. Thus $|B_k(n)| \leq |\cap B_s(n)|$. According to Lemmas VI.3.8 and VI.3.9, $|\cap B_s(n)| = 2^{2^n}$. Therefore, $|B_k(n)| \leq 2^{2^n}$.■

Next we mentioned other results which are proven in [D1] (we will not give their proof here). These properties involve only equivalence relations. In [D1], the maximal sets are C-maximal sets (or weak maximal sets), i.e. maximal sets containing [C] functions. Constant functions trivially satisfy any equivalence relation and [D2] shows that the B-maximal sets corresponding to equivalence relations are equivalent to C-maximal sets from [D1,S4]. Therefore, any C-maximal set is a B-maximal set, and then the intersection properties given in [D1] are also applied here in the context of B-maximal sets defined from equivalence relations.

Theorem VI.3.11: [D1] $|E_i(n)| = 2^{(r-t)2^{rn}} + t2^{tn}$.

Lemma VI.3.12: [D1] $|\cap E_{l(i)}(n)| = 2^{r2^n}$, for $n \geq 1$ and $l(i) = 2^r - 1 - 2^i$, $0 \leq i \leq r - 1$.

Lemma VI.3.13: [D1] $\cap E_{l(i)} = BF$, $l(i) = 2^r - 1 - 2^i$, $0 \leq i \leq r - 1$.

Lemma VI.3.14: [D1] $E_i \cap E_j \subseteq E_{i \cup j}$.

Lemma VI.3.15: [D1] $\cap E_{2i} = BF$, $0 \leq i \leq r - 1$.

Lemma VI.3.16: [D1] $E_i \cap E_j \subseteq E_{i \cap j}$.

VI.4. Classifications of set logic functions under B functions compositions

For two functions f and g of set logic we put $f \equiv g$ if for all j = 1, 2, ..., Bl(r) + k - 3 either f, $g \in B_j$ or f, $g \notin B_j$. The relation $\equiv$ is an equivalence relation on set logic functions. From Theorem VI.2.1, we can discuss the B-completeness in terms of these classes of $\equiv$ instead of

individual functions; if a set is B-complete, then replacing a function f in the set by any function in the equivalence class containing f results in another B-complete set.

We denote each possible class of functions of set logic (with respect to B-maximal sets compositions) by a binary characteristic vector $b_1b_2...b_{Bl(r)+k-3}$ where $b_j = 0$ iff $f \in B_j$, $j = 1, 2, ..., Bl(r) + k - 3$. The first $k - 2$ bits correspond to equivalence relations, and the $Bl(r) - 1$ subsequent bits correspond to central relations.

VI.4.a. Classifications over L_4 (r = 2)

In P_4 we have eight possible classes of functions, each denoted by a binary characteristic vector $b_1b_2b_3$ where $b_1 = 0$ iff $f \in E_1$, $b_2 = 0$ iff $f \in E_2$, and $b_3 = 0$ iff $f \in C_0$. The classification over L_4 can be easily done by hand-checking; we found one-place example functions f = (f(0), f(1), f(2), f(3)) in each class. However, we used the program for classification of functions (see section V.2) to find the one-place functions and also to compute the number of one-place functions in each class. The results are listed below.

Ordinal numbers	Decimal classes	Binary classes	One-place funct.	Number of funct.
1	0	000	(3, 2, 1, 0)	4
2	1	001	(2, 2, 0, 0)	12
3	2	010	(3, 2, 0, 0)	12
4	3	011	(2, 2, 1, 0)	36
5	4	100	(3, 0, 1, 0)	12
6	5	101	(2, 0, 0, 0)	36
7	6	110	(3, 0, 0, 0)	36
8	7	111	(2, 0, 0, 1)	108

Theorem VI.4.a.1: There exist 8 classes of functions of two-valued set logic under compositions with [B] functions.▪

VI.4.b. Classifications over L_8 (r = 3)

In P_8 we have 1024 possible classes of functions, each denoted by a binary characteristic vector $b_1b_2b_3b_4b_5b_6b_7b_8b_9b_{10}$ where $b_1 = 0$ iff $f \in E_1$, $b_2 = 0$ iff $f \in E_2$, $b_3 = 0$ iff $f \in E_4$, $b_4 = 0$ iff $f \in E_6$, $b_5 = 0$ iff $f \in E_5$, $b_6 = 0$ iff $f \in E_3$, $b_7 = 0$ iff $f \in C_0$, $b_8 = 0$ iff $f \in C_1$, $b_9 = 0$ iff $f \in C_2$, and $b_{10} = 0$ iff $f \in C_4$. Such order of B-maximal sets in P_8 is chosen so that the symmetric cases can be traced in an easy way. For the six B-maximal sets defined from equivalence relations, it starts with indices 1, 2, 4 which have one bit 1 in their binary representation followed by their complements, respectively. For the four subsequent B-maximal sets, it starts

with index 0 corresponding to C_0 and followed by the indices 1, 2, 4, corresponding to B-maximal sets from central relations defined from the equivalence relations having the indices 1, 2, 4, respectively (see Lemma VI.3.2). We applied the following method for classification of functions. We used the program for classification of functions (see section V.2) to generate all the one-place functions f = (f(0), f(1), f(2), f(3), f(4), f(5), f(6), f(7)) and to compute their number in each class. The computer found 200 non-empty classes that are listed here.

Ordinal numbers	Decimal classes	Binary classes	One-place funct.	Number of funct.
1	0	0000000000	(7, 6, 5, 4, 3, 2, 1, 0)	4
2	11	0000001011	(6, 6, 4, 4, 2, 2, 0, 0)	12
3	13	0000001101	(5, 4, 5, 4, 1, 0, 1, 0)	12
4	14	0000001110	(4, 4, 4, 4, 0, 0, 0, 0)	12
5	15	0000001111	(4, 4, 6, 6, 0, 0, 2, 2)	24
6	161	0010100001	(7, 6, 5, 4, 1, 0, 1, 0)	12
7	171	0010101011	(6, 6, 4, 4, 0, 0, 0, 0)	36
8	173	0010101101	(5, 4, 5, 4, 1, 0, 3, 2)	36
9	175	0010101111	(4, 4, 4, 4, 0, 0, 2, 2)	108
10	193	0011000001	(7, 6, 5, 4, 2, 2, 0, 0)	12
11	203	0011001011	(6, 6, 4, 4, 2, 3, 0, 1)	36
12	205	0011001101	(5, 4, 5, 4, 0, 0, 0, 0)	36
13	207	0011001111	(4, 4, 4, 4, 0, 1, 0, 1)	108
14	224	0011100000	(7, 6, 5, 4, 0, 0, 0, 0)	12
15	225	0011100001	(7, 6, 5, 4, 5, 5, 7, 7)	24
16	235	0011101011	(6, 6, 4, 4, 0, 1, 0, 1)	108
17	237	0011101101	(5, 4, 5, 4, 0, 0, 2, 2)	108
18	238	0011101110	(4, 4, 4, 4, 0, 1, 2, 3)	36
19	239	0011101111	(4, 4, 4, 4, 1, 0, 3, 2)	288
20	274	0100010010	(7, 6, 1, 0, 3, 2, 1, 0)	12
21	283	0100011011	(6, 6, 0, 0, 2, 2, 0, 0)	36
22	286	0100011110	(4, 4, 0, 0, 0, 0, 0, 0)	36
23	287	0100011111	(4, 4, 2, 2, 0, 0, 2, 2)	108
24	322	0101000010	(7, 6, 4, 4, 3, 2, 0, 0)	12
25	331	0101001011	(6, 6, 4, 5, 2, 2, 0, 1)	36
26	334	0101001110	(4, 4, 5, 4, 0, 0, 1, 0)	36
27	335	0101001111	(4, 4, 4, 5, 0, 0, 0, 1)	108
28	336	0101010000	(7, 6, 0, 0, 3, 2, 0, 0)	12
29	338	0101010010	(7, 6, 0, 0, 7, 6, 0, 0)	24
30	347	0101011011	(6, 6, 0, 1, 2, 2, 0, 1)	108
31	349	0101011101	(5, 4, 0, 0, 1, 0, 0, 0)	36
32	350	0101011110	(4, 4, 1, 0, 0, 0, 1, 0)	108
33	351	0101011111	(4, 4, 0, 1, 0, 0, 0, 1)	288
34	435	0110110011	(7, 6, 1, 0, 1, 0, 1, 0)	36
35	443	0110111011	(6, 6, 0, 0, 0, 0, 0, 0)	108
36	447	0110111111	(4, 4, 0, 0, 0, 0, 2, 2)	432
37	448	0111000000	(7, 6, 5, 4, 3, 2, 0, 0)	12
38	449	0111000001	(7, 6, 5, 4, 2, 2, 1, 0)	36
39	450	0111000010	(7, 6, 4, 4, 3, 2, 1, 0)	36
40	451	0111000011	(7, 6, 4, 4, 2, 2, 0, 0)	144
41	459	0111001011	(6, 6, 4, 4, 2, 2, 0, 1)	684
42	461	0111001101	(5, 4, 5, 4, 0, 0, 1, 0)	144

43	462	0111001110	(4, 4, 4, 4, 0, 0, 1, 0)	144
44	463	0111001111	(4, 4, 4, 4, 0, 0, 0, 1)	2448
45	464	0111010000	(7, 6, 0, 0, 3, 2, 1, 0)	36
46	465	0111010001	(7, 6, 0, 0, 2, 2, 0, 0)	144
47	466	0111010010	(7, 6, 0, 0, 3, 3, 0, 0)	108
48	467	0111010011	(7, 6, 0, 0, 2, 3, 0, 0)	432
49	475	0111011011	(6, 6, 0, 0, 2, 2, 0, 1)	2160
50	477	0111011101	(5, 4, 0, 0, 0, 0, 0, 0)	540
51	478	0111011110	(4, 4, 0, 0, 0, 0, 1, 0)	432
52	479	0111011111	(4, 4, 0, 0, 0, 0, 0, 1)	7668
53	480	0111100000	(7, 6, 5, 4, 0, 0, 1, 0)	36
54	481	0111100001	(7, 6, 5, 4, 1, 0, 0, 0)	108
55	482	0111100010	(7, 6, 4, 4, 0, 0, 0, 0)	144
56	483	0111100011	(7, 6, 4, 4, 1, 0, 0, 0)	432
57	491	0111101011	(6, 6, 4, 4, 0, 0, 0, 1)	2160
58	493	0111101101	(5, 4, 5, 4, 0, 0, 3, 2)	432
59	494	0111101110	(4, 4, 4, 4, 0, 0, 2, 3)	540
60	495	0111101111	(4, 4, 4, 4, 0, 0, 3, 2)	7668
61	496	0111110000	(7, 6, 0, 0, 0, 0, 0, 0)	144
62	497	0111110001	(7, 6, 0, 0, 1, 0, 0, 0)	432
63	498	0111110010	(7, 6, 0, 0, 0, 1, 0, 0)	432
64	499	0111110011	(7, 6, 0, 0, 1, 1, 0, 0)	1260
65	507	0111111011	(6, 6, 0, 0, 0, 0, 0, 1)	6804
66	509	0111111101	(5, 4, 0, 0, 0, 0, 2, 2)	1728
67	510	0111111110	(4, 4, 0, 0, 0, 0, 2, 3)	1728
68	511	0111111111	(4, 4, 0, 0, 0, 0, 3, 2)	23760
69	532	1000010100	(7, 2, 5, 0, 3, 2, 1, 0)	12
70	541	1000011101	(5, 0, 5, 0, 1, 0, 1, 0)	36
71	542	1000011110	(4, 0, 4, 0, 0, 0, 0, 0)	36
72	543	1000011111	(4, 0, 6, 2, 0, 0, 2, 2)	108
73	548	1000100100	(7, 4, 5, 4, 3, 0, 1, 0)	12
74	557	1000101101	(5, 4, 5, 6, 1, 0, 1, 2)	36
75	558	1000101110	(4, 4, 6, 4, 0, 0, 2, 0)	36
76	559	1000101111	(4, 4, 4, 6, 0, 0, 0, 2)	108
77	560	1000110000	(7, 0, 5, 0, 3, 0, 1, 0)	12
78	564	1000110100	(7, 0, 5, 0, 7, 0, 5, 0)	24
79	571	1000111011	(6, 0, 4, 0, 2, 0, 0, 0)	36
80	573	1000111101	(5, 0, 5, 2, 1, 0, 1, 2)	108
81	574	1000111110	(4, 0, 6, 0, 0, 0, 2, 0)	108
82	575	1000111111	(4, 0, 4, 2, 0, 0, 0, 2)	288
83	672	1010100000	(7, 6, 5, 4, 3, 0, 1, 0)	12
84	673	1010100001	(7, 6, 5, 4, 1, 2, 1, 0)	36
85	676	1010100100	(7, 4, 5, 4, 3, 0, 3, 0)	36
86	677	1010100101	(7, 4, 5, 4, 1, 0, 1, 0)	144
87	683	1010101011	(6, 6, 4, 4, 0, 2, 0, 0)	144
88	685	1010101101	(5, 4, 5, 4, 1, 0, 1, 2)	684
89	686	1010101110	(4, 4, 4, 4, 0, 0, 2, 0)	144
90	687	1010101111	(4, 4, 4, 4, 0, 0, 0, 2)	2448
91	688	1010110000	(7, 0, 5, 0, 3, 2, 1, 0)	36
92	689	1010110001	(7, 0, 5, 0, 1, 0, 1, 0)	144
93	692	1010110100	(7, 0, 5, 0, 3, 0, 3, 0)	108
94	693	1010110101	(7, 0, 5, 0, 1, 0, 3, 0)	432
95	699	1010111011	(6, 0, 4, 0, 0, 0, 0, 0)	540
96	701	1010111101	(5, 0, 5, 0, 1, 0, 1, 2)	2160

97	702	1010111110	(4, 0, 4, 0, 0, 0, 2, 0)	432
98	703	1010111111	(4, 0, 4, 0, 0, 0, 0, 2)	7668
99	725	1011010101	(7, 2, 5, 0, 2, 2, 0, 0)	36
100	733	1011011101	(5, 0, 5, 0, 0, 0, 0, 0)	108
101	735	1011011111	(4, 0, 4, 0, 0, 1, 0, 1)	432
102	736	1011100000	(7, 6, 5, 4, 0, 2, 0, 0)	36
103	737	1011100001	(7, 6, 5, 4, 2, 0, 0, 0)	108
104	740	1011100100	(7, 4, 5, 4, 0, 0, 0, 0)	144
105	741	1011100101	(7, 4, 5, 4, 2, 0, 0, 0)	432
106	747	1011101011	(6, 6, 4, 4, 0, 3, 0, 1)	432
107	749	1011101101	(5, 4, 5, 4, 0, 0, 0, 2)	2160
108	750	1011101110	(4, 4, 4, 4, 0, 1, 0, 3)	540
109	751	1011101111	(4, 4, 4, 4, 0, 1, 2, 1)	7668
110	752	1011110000	(7, 0, 5, 0, 0, 0, 0, 0)	144
111	753	1011110001	(7, 0, 5, 0, 1, 5, 1, 7)	432
112	756	1011110100	(7, 0, 5, 0, 0, 0, 2, 0)	432
113	757	1011110101	(7, 0, 5, 0, 1, 5, 3, 7)	1260
114	763	1011111011	(6, 0, 4, 0, 0, 1, 0, 1)	1728
115	765	1011111101	(5, 0, 5, 0, 0, 0, 0, 2)	6804
116	766	1011111110	(4, 0, 4, 0, 0, 1, 0, 3)	1728
117	767	1011111111	(4, 0, 4, 0, 0, 1, 2, 1)	23760
118	784	1100010000	(7, 6, 5, 0, 3, 2, 1, 0)	12
119	786	1100010010	(7, 6, 1, 0, 3, 6, 1, 0)	36
120	788	1100010100	(7, 2, 5, 0, 3, 2, 5, 0)	36
121	790	1100010110	(7, 2, 1, 0, 3, 2, 1, 0)	144
122	795	1100011011	(6, 6, 0, 0, 2, 6, 0, 0)	144
123	797	1100011101	(5, 0, 5, 0, 1, 0, 5, 0)	144
124	798	1100011110	(4, 0, 0, 0, 0, 0, 0, 0)	684
125	799	1100011111	(4, 0, 2, 2, 0, 0, 2, 2)	2448
126	816	1100110000	(7, 0, 5, 0, 7, 0, 1, 0)	36
127	818	1100110010	(7, 0, 1, 0, 3, 0, 1, 0)	144
128	820	1100110100	(7, 0, 5, 0, 3, 0, 5, 0)	108
129	822	1100110110	(7, 0, 1, 0, 3, 0, 5, 0)	432
130	827	1100111011	(6, 0, 0, 0, 2, 0, 0, 0)	540
131	829	1100111101	(5, 0, 5, 2, 1, 0, 5, 2)	432
132	830	1100111110	(4, 0, 2, 0, 0, 0, 2, 0)	2160
133	831	1100111111	(4, 0, 0, 2, 0, 0, 0, 2)	7668
134	848	1101010000	(7, 6, 0, 0, 7, 2, 0, 0)	36
135	850	1101010010	(7, 6, 0, 0, 3, 6, 0, 0)	108
136	852	1101010100	(7, 2, 0, 0, 3, 2, 0, 0)	144
137	854	1101010110	(7, 2, 0, 0, 3, 6, 0, 0)	432
138	859	1101011011	(6, 6, 0, 1, 2, 6, 0, 1)	432
139	861	1101011101	(5, 0, 0, 0, 1, 0, 0, 0)	540
140	862	1101011110	(4, 0, 1, 0, 0, 0, 1, 0)	2160
141	863	1101011111	(4, 0, 0, 1, 0, 0, 0, 1)	7668
142	870	1101100110	(7, 4, 4, 4, 3, 0, 0, 0)	36
143	878	1101101110	(4, 4, 4, 7, 0, 0, 0, 3)	108
144	879	1101101111	(4, 4, 5, 6, 0, 0, 1, 2)	432
145	880	1101110000	(7, 0, 0, 0, 3, 0, 0, 0)	144
146	882	1101110010	(7, 0, 0, 0, 3, 4, 0, 0)	432
147	884	1101110100	(7, 0, 0, 0, 3, 0, 4, 0)	432
148	886	1101110110	(7, 0, 0, 0, 3, 4, 4, 0)	1260
149	891	1101111011	(6, 0, 0, 1, 2, 0, 0, 1)	1728
150	893	1101111101	(5, 0, 0, 2, 1, 0, 0, 2)	1728

151	894	1101111110	(4, 0, 0, 3, 0, 0, 0, 3)	6804
152	895	1101111111	(4, 0, 1, 2, 0, 0, 1, 2)	23760
153	944	1110110000	(7, 0, 5, 0, 3, 0, 7, 0)	144
154	945	1110110001	(7, 0, 5, 0, 1, 0, 7, 0)	576
155	946	1110110010	(7, 0, 1, 0, 3, 0, 7, 0)	576
156	947	1110110011	(7, 0, 1, 0, 1, 0, 1, 0)	2268
157	948	1110110100	(7, 0, 5, 0, 3, 2, 5, 0)	432
158	949	1110110101	(7, 0, 5, 0, 1, 0, 5, 0)	1728
159	950	1110110110	(7, 0, 1, 0, 3, 0, 3, 0)	1728
160	951	1110110111	(7, 0, 1, 0, 1, 0, 3, 0)	6912
161	955	1110111011	(6, 0, 0, 0, 0, 0, 0, 0)	10692
162	957	1110111101	(5, 0, 5, 0, 1, 0, 5, 2)	8640
163	958	1110111110	(4, 0, 0, 0, 0, 0, 2, 0)	8640
164	959	1110111111	(4, 0, 0, 0, 0, 0, 0, 2)	187488
165	976	1111010000	(7, 6, 0, 0, 3, 7, 0, 0)	144
166	977	1111010001	(7, 6, 0, 0, 2, 7, 0, 0)	576
167	978	1111010010	(7, 6, 0, 0, 3, 6, 1, 0)	432
168	979	1111010011	(7, 6, 0, 0, 2, 6, 0, 0)	1728
169	980	1111010100	(7, 2, 0, 0, 3, 2, 1, 0)	576
170	981	1111010101	(7, 2, 0, 0, 2, 2, 0, 0)	2268
171	982	1111010110	(7, 2, 0, 0, 3, 3, 0, 0)	1728
172	983	1111010111	(7, 2, 0, 0, 2, 3, 0, 0)	6912
173	987	1111011011	(6, 6, 0, 0, 2, 6, 0, 1)	8640
174	989	1111011101	(5, 0, 0, 0, 0, 0, 0, 0)	10692
175	990	1111011110	(4, 0, 0, 0, 0, 0, 1, 0)	8640
176	991	1111011111	(4, 0, 0, 0, 0, 0, 0, 1)	187488
177	992	1111100000	(7, 6, 5, 4, 0, 2, 1, 0)	144
178	993	1111100001	(7, 6, 5, 4, 1, 2, 0, 0)	432
179	994	1111100010	(7, 6, 4, 4, 0, 2, 0, 0)	576
180	995	1111100011	(7, 6, 4, 4, 1, 2, 0, 0)	1728
181	996	1111100100	(7, 4, 5, 4, 0, 0, 1, 0)	576
182	997	1111100101	(7, 4, 5, 4, 1, 0, 0, 0)	1728
183	998	1111100110	(7, 4, 4, 4, 0, 0, 0, 0)	2268
184	999	1111100111	(7, 4, 4, 4, 1, 0, 0, 0)	6912
185	1003	1111101011	(6, 6, 4, 4, 0, 2, 0, 1)	8640
186	1005	1111101101	(5, 4, 5, 4, 0, 0, 1, 2)	8640
187	1006	1111101110	(4, 4, 4, 4, 0, 0, 0. 3)	10692
188	1007	1111101111	(4, 4, 4, 4, 0, 0, 1, 2)	187488
189	1008	1111110000	(7, 0, 0, 0, 0, 0, 0, 0)	15228
190	1009	1111110001	(7, 0, 0, 0, 1, 0, 0, 0)	46080
191	1010	1111110010	(7, 0, 0, 0, 0, 1, 0, 0)	46080
192	1011	1111110011	(7, 0, 0, 0, 1, 1, 0, 0)	139428
193	1012	1111110100	(7, 0, 0, 0, 0, 0, 2, 0)	46080
194	1013	1111110101	(7, 0, 0, 0, 1, 0, 2, 0)	139428
195	1014	1111110110	(7, 0, 0, 0, 0, 1, 2, 0)	139428
196	1015	1111110111	(7, 0, 0, 0, 1, 1, 2, 0)	421632
197	1019	1111111011	(6, 0, 0, 0, 0, 0, 0, 1)	740448
198	1021	1111111101	(5, 0, 0, 0, 0, 0, 0, 2)	740448
199	1022	1111111110	(4, 0, 0, 0, 0, 0, 0, 3)	740448
200	1023	1111111111	(4, 0, 0, 0, 0, 0, 1, 2)	12679416

The remaining classes are all empty. This can be seen by examining the intersection properties of B-maximal sets, i.e. by applying certains lemmas from section IV.3. Below we describe and list the empty classes from each applicable lemma.

From Lemmas VI.3.2 and VI.3.3 it follows that classes such that (b_1, b_7, b_8) or (b_2, b_7, b_9) or (b_3, b_7, b_{10}) = (0, 0, 1) are empty (ex. class 323 = 0101000011 is empty). There are 296 such classes:

1=0000000001, 2=0000000010, 3=0000000011, 4=0000000100, 5=0000000101, 6=0000000110, 7=0000000111, 17=0000010001, 18=0000010010, 19=0000010011, 20=0000010100, 21=0000010101, 22=0000010110, 23=0000010111, 33=0000100001, 34=0000100010, 35=0000100011, 36=0000100100, 37=0000100101, 38=0000100110, 39=0000100111, 49=0000110001, 50=0000110010, 51=0000110011, 52=0000110100, 53=0000110101, 54=0000110110, 55=0000110111, 65=0001000001, 66=0001000010, 67=0001000011, 68=0001000100, 69=0001000101, 70=0001000110, 71=0001000111, 81=0001010001, 82=0001010010, 83=0001010011, 84=0001010100, 85=0001010101, 86=0001010110, 87=0001010111, 97=0001100001, 98=0001100010, 99=0001100011, 100=0001100100, 101=0001100101, 102=0001100110, 103=0001100111, 113=0001110001, 114=0001110010, 115=0001110011, 116=0001110100, 117=0001110101, 118=0001110110, 119=0001110111, 130=0010000010, 131=0010000011, 132=0010000100, 133=0010000101, 134=0010000110, 135=0010000111, 146=0010010010, 147=0010010011, 148=0010010100, 149=0010010101, 150=0010010110, 151=0010010111, 162=0010100010, 163=0010100011, 164=0010100100, 165=0010100101, 166=0010100110, 167=0010100111, 178=0010110010, 179=0010110011, 180=0010110100, 181=0010110101, 182=0010110110, 183=0010110111, 194=0011000010, 195=0011000011, 196=0011000100, 197=0011000101, 198=0011000110, 199=0011000111, 210=0011010010, 211=0011010011, 212=0011010100, 213=0011010101, 214=0011010110, 215=0011010111, 226=0011100010, 227=0011100011, 228=0011100100, 229=0011100101, 230=0011100110, 231=0011100111, 242=0011110010, 243=0011110011, 244=0011110100, 245=0011110101, 246=0011110110, 247=0011110111, 257=0100000001, 259=0100000011, 260=0100000100, 261=0100000101, 262=0100000110, 263=0100000111, 273=0100010001, 275=0100010011, 276=0100010100, 277=0100010101, 278=0100010110, 279=0100010111, 289=0100100001, 291=0100100011, 292=0100100100, 293=0100100101, 294=0100100110, 295=0100100111, 305=0100110001, 307=0100110011, 308=0100110100, 309=0100110101, 310=0100110110, 311=0100110111, 321=0101000001, 323=0101000011, 324=0101000100, 325=0101000101, 326=0101000110, 327=0101000111, 337=0101010001, 339=0101010011, 340=0101010100, 341=0101010101, 342=0101010110, 343=0101010111, 353=0101100001, 355=0101100011, 356=0101100100, 357=0101100101, 358=0101100110, 359=0101100111, 369=0101110001, 371=0101110011, 372=0101110100, 373=0101110101, 374=0101110110, 375=0101110111, 388=0110000100, 389=0110000101, 390=0110000110, 391=0110000111, 404=0110010100, 405=0110010101, 406=0110010110, 407=0110010111, 420=0110100100, 421=0110100101, 422=0110100110, 423=0110100111, 436=0110110100, 437=0110110101, 438=0110110110, 439=0110110111, 452=0111000100, 453=0111000101, 454=0111000110, 455=0111000111, 468=0111010100, 469=0111010101, 470=0111010110, 471=0111010111, 484=0111100100, 485=0111100101, 486=0111100110, 487=0111100111, 500=0111110100, 501=0111110101, 502=0111110110, 503=0111110111, 513=1000000001, 514=1000000010, 515=1000000011, 517=1000000101, 518=1000000110, 519=1000000111, 529=1000010001, 530=1000010010, 531=1000010011, 533=1000010101, 534=1000010110, 535=1000010111, 545=1000100001, 546=1000100010, 547=1000100011, 549=1000100101, 550=1000100110, 551=1000100111, 561=1000110001, 562=1000110010, 563=1000110011, 565=1000110101, 566=1000110110, 567=1000110111, 577=1001000001, 578=1001000010, 579=1001000011, 581=1001000101, 582=1001000110, 583=1001000111, 593=1001010001, 594=1001010010, 595=1001010011, 597=1001010101, 598=1001010110, 599=1001010111, 609=1001100001, 610=1001100010, 611=1001100011, 613=1001100101, 614=1001100110, 615=1001100111, 625=1001110001, 626=1001110010, 627=1001110011, 629=1001110101, 630=1001110110, 631=1001110111, 642=1010000010, 643=1010000011, 646=1010000110, 647=1010000111, 658=1010010010, 659=1010010011, 662=1010010110, 663=1010010111, 674=1010100010, 675=1010100011, 678=1010100110, 679=1010100111, 690=1010110010, 691=1010110011, 694=1010110110, 695=1010110111, 706=1011000010, 707=1011000011, 710=1011000110, 711=1011000111, 722=1011010010, 723=1011010011, 726=1011010110, 727=1011010111, 738=1011100010, 739=1011100011, 742=1011100110, 743=1011100111, 754=1011110010, 755=1011110011, 758=1011110110, 759=1011110111, 769=1100000001, 771=1100000011, 773=1100000101, 775=1100000111, 785=1100010001, 787=1100010011, 789=1100010101, 791=1100010111, 801=1100100001, 803=1100100011, 805=1100100101, 807=1100100111, 817=1100110001, 819=1100110011, 821=1100110101, 823=1100110111, 833=1101000001, 835=1101000011, 837=1101000101, 839=1101000111, 849=1101010001, 851=1101010011, 853=1101010101, 855=1101010111, 865=1101100001, 867=1101100011, 869=1101100101, 871=1101100111, 881=1101110001, 883=1101110011, 885=1101110101, 887=1101110111.

From Lemma VI.3.4 it follows that classes such that (b_1, b_4, b_j, b_{j-7}) or (b_2, b_5, b_j, b_{j-7}) or (b_3, b_6, b_j, b_{j-7}) = (0, 0, 0, 1), $8 \leq j \leq 10$, are empty (ex. class 721 = **1011010001** is empty). There are 288 such classes:

128=0010000000, 130=0010000010, 132=0010000100, 134=0010000110, 136=0010001000, 138=0010001010, 140=0010001100,
142=0010001110, 144=0010010000, 146=0010010010, 148=0010010100, 150=0010010110, 152=0010011000, 154=0010011010,
156=0010011100, 158=0010011110, 160=0010100000, 162=0010100010, 164=0010100100, 166=0010100110, 168=0010101000,
170=0010101010, 172=0010101100, 174=0010101110, 176=0010110000, 178=0010110010, 180=0010110100, 182=0010110110,
184=0010111000, 186=0010111010, 188=0010111100, 190=0010111110, 192=0011000000, 194=0011000010, 196=0011000100,
198=0011000110, 200=0011001000, 202=0011001010, 204=0011001100, 206=0011001110, 208=0011010000, 210=0011010010,
212=0011010100, 214=0011010110, 216=0011011000, 218=0011011010, 220=0011011100, 222=0011011110, 256=0100000000,
257=0100000001, 260=0100000100, 261=0100000101, 264=0100001000, 265=0100001001, 268=0100001100, 269=0100001101,
272=0100010000, 273=0100010001, 276=0100010100, 277=0100010101, 280=0100011000, 281=0100011001, 284=0100011100,
285=0100011101, 288=0100100000, 289=0100100001, 292=0100100100, 293=0100100101, 296=0100101000, 297=0100101001,
300=0100101100, 301=0100101101, 304=0100110000, 305=0100110001, 308=0100110100, 309=0100110101, 312=0100111000,
313=0100111001, 316=0100111100, 317=0100111101, 320=0101000000, 321=0101000001, 324=0101000100, 325=0101000101,
328=0101001000, 329=0101001001, 332=0101001100, 333=0101001101, 352=0101100000, 353=0101100001, 356=0101100100,
357=0101100101, 360=0101101000, 361=0101101001, 364=0101101100, 365=0101101101, 384=0110000000, 385=0110000001,
386=0110000010, 388=0110000100, 389=0110000101, 390=0110000110, 392=0110001000, 393=0110001001, 394=0110001010,
396=0110001100, 397=0110001101, 398=0110001110, 400=0110010000, 401=0110010001, 402=0110010010, 404=0110010100,
405=0110010101, 406=0110010110, 408=0110011000, 409=0110011001, 410=0110011010, 412=0110011100, 413=0110011101,
414=0110011110, 416=0110100000, 417=0110100001, 418=0110100010, 420=0110100100, 421=0110100101, 422=0110100110,
424=0110101000, 425=0110101001, 426=0110101010, 428=0110101100, 429=0110101101, 430=0110101110, 432=0110110000,
433=0110110001, 434=0110110010, 436=0110110100, 437=0110110101, 438=0110110110, 440=0110111000, 441=0110111001,
442=0110111010, 444=0110111100, 445=0110111101, 446=0110111110, 512=1000000000, 513=1000000001, 514=1000000010,
515=1000000011, 520=1000001000, 521=1000001001, 522=1000001010, 523=1000001011, 528=1000010000, 529=1000010001,
530=1000010010, 531=1000010011, 536=1000011000, 537=1000011001, 538=1000011010, 539=1000011011, 544=1000100000,
545=1000100001, 546=1000100010, 547=1000100011, 552=1000101000, 553=1000101001, 554=1000101010, 555=1000101011,
576=1001000000, 577=1001000001, 578=1001000010, 579=1001000011, 584=1001001000, 585=1001001001, 586=1001001010,
587=1001001011, 592=1001010000, 593=1001010001, 594=1001010010, 595=1001010011, 600=1001011000, 601=1001011001,
602=1001011010, 603=1001011011, 608=1001100000, 609=1001100001, 610=1001100010, 611=1001100011, 616=1001101000,
617=1001101001, 618=1001101010, 619=1001101011, 640=1010000000, 641=1010000001, 642=1010000010, 643=1010000011,
644=1010000100, 646=1010000110, 648=1010001000, 649=1010001001, 650=1010001010, 651=1010001011, 652=1010001100,
654=1010001110, 656=1010010000, 657=1010010001, 658=1010010010, 659=1010010011, 660=1010010100, 662=1010010110,
664=1010011000, 665=1010011001, 666=1010011010, 667=1010011011, 668=1010011100, 670=1010011110, 704=1011000000,
705=1011000001, 706=1011000010, 707=1011000011, 708=1011000100, 710=1011000110, 712=1011001000, 713=1011001001,
714=1011001010, 715=1011001011, 716=1011001100, 718=1011001110, 720=1011010000, 721=1011010001, 722=1011010010,
723=1011010011, 724=1011010100, 726=1011010110, 728=1011011000, 729=1011011001, 730=1011011010, 731=1011011011,
732=1011011100, 734=1011011110, 768=1100000000, 769=1100000001, 770=1100000010, 771=1100000011, 772=1100000100,
773=1100000101, 776=1100001000, 777=1100001001, 778=1100001010, 779=1100001011, 780=1100001100, 781=1100001101,
800=1100100000, 801=1100100001, 802=1100100010, 803=1100100011, 804=1100100100, 805=1100100101, 808=1100101000,
809=1100101001, 810=1100101010, 811=1100101011, 812=1100101100, 813=1100101101, 832=1101000000, 833=1101000001,
834=1101000010, 835=1101000011, 836=1101000100, 837=1101000101, 840=1101001000, 841=1101001001, 842=1101001010,
843=1101001011, 844=1101001100, 845=1101001101, 864=1101100000, 865=1101100001, 866=1101100010, 867=1101100011,
868=1101100100, 869=1101100101, 872=1101101000, 873=1101101001, 874=1101101010, 875=1101101011, 876=1101101100,
877=1101101101.

From Lemmas VI.3.6 and VI.3.7 it follows that classes such that (b_8, b_9, b_7) or (b_8, b_{10}, b_7) or (b_9, b_{10}, b_7) = (0, 0, 1) are empty (ex. class 508 = 011111**100** is empty). There are 256 such classes:

8=0000001000, 9=0000001001, 10=0000001010, 12=0000001100, 24=0000011000, 25=0000011001, 26=0000011010, 28=0000011100,
40=0000101000, 41=0000101001, 42=0000101010, 44=0000101100, 56=0000111000, 57=0000111001, 58=0000111010, 60=0000111100,
72=0001001000, 73=0001001001, 74=0001001010, 76=0001001100, 88=0001011000, 89=0001011001, 90=0001011010, 92=0001011100,
104=0001101000, 105=0001101001, 106=0001101010, 108=0001101100, 120=0001111000, 121=0001111001, 122=0001111010,
124=0001111100, 136=0010001000, 137=0010001001, 138=0010001010, 140=0010001100, 152=0010011000, 153=0010011001,
154=0010011010, 156=0010011100, 168=0010101000, 169=0010101001, 170=0010101010, 172=0010101100, 184=0010111000,
185=0010111001, 186=0010111010, 188=0010111100, 200=0011001000, 201=0011001001, 202=0011001010, 204=0011001100,
216=0011011000, 217=0011011001, 218=0011011010, 220=0011011100, 232=0011101000, 233=0011101001, 234=0011101010,
236=0011101100, 248=0011111000, 249=0011111001, 250=0011111010, 252=0011111100, 264=0100001000, 265=0100001001,
266=0100001010, 268=0100001100, 280=0100011000, 281=0100011001, 282=0100011010, 284=0100011100, 296=0100101000,
297=0100101001, 298=0100101010, 300=0100101100, 312=0100111000, 313=0100111001, 314=0100111010, 316=0100111100,
328=0101001000, 329=0101001001, 330=0101001010, 332=0101001100, 344=0101011000, 345=0101011001, 346=0101011010,
348=0101011100, 360=0101101000, 361=0101101001, 362=0101101010, 364=0101101100, 376=0101111000, 377=0101111001,
378=0101111010, 380=0101111100, 392=0110001000, 393=0110001001, 394=0110001010, 396=0110001100, 408=0110011000,
409=0110011001, 410=0110011010, 412=0110011100, 424=0110101000, 425=0110101001, 426=0110101010, 428=0110101100,
440=0110111000, 441=0110111001, 442=0110111010, 444=0110111100, 456=0111001000, 457=0111001001, 458=0111001010,
460=0111001100, 472=0111011000, 473=0111011001, 474=0111011010, 476=0111011100, 488=0111101000, 489=0111101001,
490=0111101010, 492=0111101100, 504=0111111000, 505=0111111001, 506=0111111010, 508=0111111100, 520=1000001000,
521=1000001001, 522=1000001010, 524=1000001100, 536=1000011000, 537=1000011001, 538=1000011010, 540=1000011100,
552=1000101000, 553=1000101001, 554=1000101010, 556=1000101100, 568=1000111000, 569=1000111001, 570=1000111010,
572=1000111100, 584=1001001000, 585=1001001001, 586=1001001010, 588=1001001100, 600=1001011000, 601=1001011001,
602=1001011010, 604=1001011100, 616=1001101000, 617=1001101001, 618=1001101010, 620=1001101100, 632=1001111000,
633=1001111001, 634=1001111010, 636=1001111100, 648=1010001000, 649=1010001001, 650=1010001010, 652=1010001100,
664=1010011000, 665=1010011001, 666=1010011010, 668=1010011100, 680=1010101000, 681=1010101001, 682=1010101010,
684=1010101100, 696=1010111000, 697=1010111001, 698=1010111010, 700=1010111100, 712=1011001000, 713=1011001001,
714=1011001010, 716=1011001100, 728=1011011000, 729=1011011001, 730=1011011010, 732=1011011100, 744=1011101000,
745=1011101001, 746=1011101010, 748=1011101100, 760=1011111000, 761=1011111001, 762=1011111010, 764=1011111100,
776=1100001000, 777=1100001001, 778=1100001010, 780=1100001100, 792=1100011000, 793=1100011001, 794=1100011010,
796=1100011100, 808=1100101000, 809=1100101001, 810=1100101010, 812=1100101100, 824=1100111000, 825=1100111001,
826=1100111010, 828=1100111100, 840=1101001000, 841=1101001001, 842=1101001010, 844=1101001100, 856=1101011000,
857=1101011001, 858=1101011010, 860=1101011100, 872=1101101000, 873=1101101001, 874=1101101010, 876=1101101100,
888=1101111000, 889=1101111001, 890=1101111010, 892=1101111100, 904=1110001000, 905=1110001001, 906=1110001010,
908=1110001100, 920=1110011000, 921=1110011001, 922=1110011010, 924=1110011100, 936=1110101000, 937=1110101001,
938=1110101010, 940=1110101100, 952=1110111000, 953=1110111001, 954=1110111010, 956=1110111100, 968=1111001000,
969=1111001001, 970=1111001010, 972=1111001100, 984=1111011000, 985=1111011001, 986=1111011010, 988=1111011100,
1000=1111101000, 1001=1111101001, 1002=1111101010, 1004=1111101100, 1016=1111111000, 1017=1111111001, 1018=1111111010,
1020=1111111100.

From Lemma VI.3.13 it follows that classes such that $(b_4, b_5, b_6, b_j) = (0, 0, 0, 1)$, $1 \le j \le 3$, are empty (ex. class 129 = 00**10000001** is empty). There are 112 such classes:

128=0010000000, 129=0010000001, 130=0010000010, 131=0010000011, 132=0010000100, 133=0010000101, 134=0010000110,
135=0010000111, 136=0010001000, 137=0010001001, 138=0010001010, 139=0010001011, 140=0010001100, 141=0010001101,
142=0010001110, 143=0010001111, 256=0100000000, 257=0100000001, 258=0100000010, 259=0100000011, 260=0100000100,
261=0100000101, 262=0100000110, 263=0100000111, 264=0100001000, 265=0100001001, 266=0100001010, 267=0100001011,
268=0100001100, 269=0100001101, 270=0100001110, 271=0100001111, 384=0110000000, 385=0110000001, 386=0110000010,
387=0110000011, 388=0110000100, 389=0110000101, 390=0110000110, 391=0110000111, 392=0110001000, 393=0110001001,
394=0110001010, 395=0110001011, 396=0110001100, 397=0110001101, 398=0110001110, 399=0110001111, 512=1000000000,
513=1000000001, 514=1000000010, 515=1000000011, 516=1000000100, 517=1000000101, 518=1000000110, 519=1000000111,
520=1000001000, 521=1000001001, 522=1000001010, 523=1000001011, 524=1000001100, 525=1000001101, 526=1000001110,
527=1000001111, 640=1010000000, 641=1010000001, 642=1010000010, 643=1010000011, 644=1010000100, 645=1010000101,
646=1010000110, 647=1010000111, 648=1010001000, 649=1010001001, 650=1010001010, 651=1010001011, 652=1010001100,
653=1010001101, 654=1010001110, 655=1010001111, 768=1100000000, 769=1100000001, 770=1100000010, 771=1100000011,

772=1100000100, 773=1100000101, 774=1100000110, 775=1100000111, 776=1100001000, 777=1100001001, 778=1100001010, 779=1100001011, 780=1100001100, 781=1100001101, 782=1100001110, 783=1100001111, 896=1110000000, 897=1110000001, 898=1110000010, 899=1110000011, 900=1110000100, 901=1110000101, 902=1110000110, 903=1110000111, 904=1110001000, 905=1110001001, 906=1110001010, 907=1110001011, 908=1110001100, 909=1110001101, 910=1110001110, 911=1110001111.

From Lemma VI.3.14 it follows that classes such that (b_1, b_2, b_6) or (b_1, b_3, b_5) or (b_2, b_3, b_4) = (0, 0, 1) are empty (ex. class 624 = 1001110000 is empty). There are 304 such classes:

16=0000010000, 17=0000010001, 18=0000010010, 19=0000010011, 20=0000010100, 21=0000010101, 22=0000010110, 23=0000010111, 24=0000011000, 25=0000011001, 26=0000011010, 27=0000011011, 28=0000011100, 29=0000011101, 30=0000011110, 31=0000011111, 32=0000100000, 33=0000100001, 34=0000100010, 35=0000100011, 36=0000100100, 37=0000100101, 38=0000100110, 39=0000100111, 40=0000101000, 41=0000101001, 42=0000101010, 43=0000101011, 44=0000101100, 45=0000101101, 46=0000101110, 47=0000101111, 48=0000110000, 49=0000110001, 50=0000110010, 51=0000110011, 52=0000110100, 53=0000110101, 54=0000110110, 55=0000110111, 56=0000111000, 57=0000111001, 58=0000111010, 59=0000111011, 60=0000111100, 61=0000111101, 62=0000111110, 63=0000111111, 64=0001000000, 65=0001000001, 66=0001000010, 67=0001000011, 68=0001000100, 69=0001000101, 70=0001000110, 71=0001000111, 72=0001001000, 73=0001001001, 74=0001001010, 75=0001001011, 76=0001001100, 77=0001001101, 78=0001001110, 79=0001001111, 80=0001010000, 81=0001010001, 82=0001010010, 83=0001010011, 84=0001010100, 85=0001010101, 86=0001010110, 87=0001010111, 88=0001011000, 89=0001011001, 90=0001011010, 91=0001011011, 92=0001011100, 93=0001011101, 94=0001011110, 95=0001011111, 96=0001100000, 97=0001100001, 98=0001100010, 99=0001100011, 100=0001100100, 101=0001100101, 102=0001100110, 103=0001100111, 104=0001101000, 105=0001101001, 106=0001101010, 107=0001101011, 108=0001101100, 109=0001101101, 110=0001101110, 111=0001101111, 112=0001110000, 113=0001110001, 114=0001110010, 115=0001110011, 116=0001110100, 117=0001110101, 118=0001110110, 119=0001110111, 120=0001111000, 121=0001111001, 122=0001111010, 123=0001111011, 124=0001111100, 125=0001111101, 126=0001111110, 127=0001111111, 144=0010010000, 145=0010010001, 146=0010010010, 147=0010010011, 148=0010010100, 149=0010010101, 150=0010010110, 151=0010010111, 152=0010011000, 153=0010011001, 154=0010011010, 155=0010011011, 156=0010011100, 157=0010011101, 158=0010011110, 159=0010011111, 176=0010110000, 177=0010110001, 178=0010110010, 179=0010110011, 180=0010110100, 181=0010110101, 182=0010110110, 183=0010110111, 184=0010111000, 185=0010111001, 186=0010111010, 187=0010111011, 188=0010111100, 189=0010111101, 190=0010111110, 191=0010111111, 208=0011010000, 209=0011010001, 210=0011010010, 211=0011010011, 212=0011010100, 213=0011010101, 214=0011010110, 215=0011010111, 216=0011011000, 217=0011011001, 218=0011011010, 219=0011011011, 220=0011011100, 221=0011011101, 222=0011011110, 223=0011011111, 240=0011110000, 241=0011110001, 242=0011110010, 243=0011110011, 244=0011110100, 245=0011110101, 246=0011110110, 247=0011110111, 248=0011111000, 249=0011111001, 250=0011111010, 251=0011111011, 252=0011111100, 253=0011111101, 254=0011111110, 255=0011111111, 288=0100100000, 289=0100100001, 290=0100100010, 291=0100100011, 292=0100100100, 293=0100100101, 294=0100100110, 295=0100100111, 296=0100101000, 297=0100101001, 298=0100101010, 299=0100101011, 300=0100101100, 301=0100101101, 302=0100101110, 303=0100101111, 304=0100110000, 305=0100110001, 306=0100110010, 307=0100110011, 308=0100110100, 309=0100110101, 310=0100110110, 311=0100110111, 312=0100111000, 313=0100111001, 314=0100111010, 315=0100111011, 316=0100111100, 317=0100111101, 318=0100111110, 319=0100111111, 352=0101100000, 353=0101100001, 354=0101100010, 355=0101100011, 356=0101100100, 357=0101100101, 358=0101100110, 359=0101100111, 360=0101101000, 361=0101101001, 362=0101101010, 363=0101101011, 364=0101101100, 365=0101101101, 366=0101101110, 367=0101101111, 368=0101110000, 369=0101110001, 370=0101110010, 371=0101110011, 372=0101110100, 373=0101110101, 374=0101110110, 375=0101110111, 376=0101111000, 377=0101111001, 378=0101111010, 379=0101111011, 380=0101111100, 381=0101111101, 382=0101111110, 383=0101111111, 576=1001000000, 577=1001000001, 578=1001000010, 579=1001000011, 580=1001000100, 581=1001000101, 582=1001000110, 583=1001000111, 584=1001001000, 585=1001001001, 586=1001001010, 587=1001001011, 588=1001001100, 589=1001001101, 590=1001001110, 591=1001001111, 592=1001010000, 593=1001010001, 594=1001010010, 595=1001010011, 596=1001010100, 597=1001010101, 598=1001010110, 599=1001010111, 600=1001011000, 601=1001011001, 602=1001011010, 603=1001011011, 604=1001011100, 605=1001011101, 606=1001011110, 607=1001011111, 608=1001100000, 609=1001100001, 610=1001100010, 611=1001100011, 612=1001100100, 613=1001100101, 614=1001100110, 615=1001100111, 616=1001101000, 617=1001101001, 618=1001101010, 619=1001101011, 620=1001101100, 621=1001101101, 622=1001101110, 623=1001101111, 624=1001110000, 625=1001110001, 626=1001110010, 627=1001110011, 628=1001110100, 629=1001110101, 630=1001110110, 631=1001110111, 632=1001111000, 633=1001111001, 634=1001111010, 635=1001111011, 636=1001111100, 637=1001111101, 638=1001111110, 639=1001111111.

From Lemma VI.3.15 it follows that classes such that $(b_1, b_2, b_3, b_j) = (0, 0, 0, 1)$, $4 \le j \le 6$, are empty (ex. class 127 = **0001111**111 is empty). There are 112 such classes:

16=0000010000, 17=0000010001, 18=0000010010, 19=0000010011, 20=0000010100, 21=0000010101, 22=0000010110, 23=0000010111, 24=0000011000, 25=0000011001, 26=0000011010, 27=0000011011, 28=0000011100, 29=0000011101, 30=0000011110, 31=0000011111, 32=0000100000, 33=0000100001, 34=0000100010, 35=0000100011, 36=0000100100, 37=0000100101, 38=0000100110, 39=0000100111, 40=0000101000, 41=0000101001, 42=0000101010, 43=0000101011, 44=0000101100, 45=0000101101, 46=0000101110, 47=0000101111, 48=0000110000, 49=0000110001, 50=0000110010, 51=0000110011, 52=0000110100, 53=0000110101, 54=0000110110, 55=0000110111, 56=0000111000, 57=0000111001, 58=0000111010, 59=0000111011, 60=0000111100, 61=0000111101, 62=0000111110, 63=0000111111, 64=0001000000, 65=0001000001, 66=0001000010, 67=0001000011, 68=0001000100, 69=0001000101, 70=0001000110, 71=0001000111, 72=0001001000, 73=0001001001, 74=0001001010, 75=0001001011, 76=0001001100, 77=0001001101, 78=0001001110, 79=0001001111, 80=0001010000, 81=0001010001, 82=0001010010, 83=0001010011, 84=0001010100, 85=0001010101, 86=0001010110, 87=0001010111, 88=0001011000, 89=0001011001, 90=0001011010, 91=0001011011, 92=0001011100, 93=0001011101, 94=0001011110, 95=0001011111, 96=0001100000, 97=0001100001, 98=0001100010, 99=0001100011, 100=0001100100, 101=0001100101, 102=0001100110, 103=0001100111, 104=0001101000, 105=0001101001, 106=0001101010, 107=0001101011, 108=0001101100, 109=0001101101, 110=0001101110, 111=0001101111, 112=0001110000, 113=0001110001, 114=0001110010, 115=0001110011, 116=0001110100, 117=0001110101, 118=0001110110, 119=0001110111, 120=0001111000, 121=0001111001, 122=0001111010, 123=0001111011, 124=0001111100, 125=0001111101, 126=0001111110, 127=0001111111.

From Lemma VI.3.16 it follows that classes such that (b_4, b_5, b_3) or (b_4, b_6, b_2) or $(b_5, b_6, b_1) = (0, 0, 1)$ are empty (ex. class 975 = 1111**001**111 is empty). There are 304 such classes:

128=0010000000, 129=0010000001, 130=0010000010, 131=0010000011, 132=0010000100, 133=0010000101, 134=0010000110, 135=0010000111, 136=0010001000, 137=0010001001, 138=0010001010, 139=0010001011, 140=0010001100, 141=0010001101, 142=0010001110, 143=0010001111, 144=0010010000, 145=0010010001, 146=0010010010, 147=0010010011, 148=0010010100, 149=0010010101, 150=0010010110, 151=0010010111, 152=0010011000, 153=0010011001, 154=0010011010, 155=0010011011, 156=0010011100, 157=0010011101, 158=0010011110, 159=0010011111, 256=0100000000, 257=0100000001, 258=0100000010, 259=0100000011, 260=0100000100, 261=0100000101, 262=0100000110, 263=0100000111, 264=0100001000, 265=0100001001, 266=0100001010, 267=0100001011, 268=0100001100, 269=0100001101, 270=0100001110, 271=0100001111, 288=0100100000, 289=0100100001, 290=0100100010, 291=0100100011, 292=0100100100, 293=0100100101, 294=0100100110, 295=0100100111, 296=0100101000, 297=0100101001, 298=0100101010, 299=0100101011, 300=0100101100, 301=0100101101, 302=0100101110, 303=0100101111, 384=0110000000, 385=0110000001, 386=0110000010, 387=0110000011, 388=0110000100, 389=0110000101, 390=0110000110, 391=0110000111, 392=0110001000, 393=0110001001, 394=0110001010, 395=0110001011, 396=0110001100, 397=0110001101, 398=0110001110, 399=0110001111, 400=0110010000, 401=0110010001, 402=0110010010, 403=0110010011, 404=0110010100, 405=0110010101, 406=0110010110, 407=0110010111, 408=0110011000, 409=0110011001, 410=0110011010, 411=0110011011, 412=0110011100, 413=0110011101, 414=0110011110, 415=0110011111, 416=0110100000, 417=0110100001, 418=0110100010, 419=0110100011, 420=0110100100, 421=0110100101, 422=0110100110, 423=0110100111, 424=0110101000, 425=0110101001, 426=0110101010, 427=0110101011, 428=0110101100, 429=0110101101, 430=0110101110, 431=0110101111, 512=1000000000, 513=1000000001, 514=1000000010, 515=1000000011, 516=1000000100, 517=1000000101, 518=1000000110, 519=1000000111, 520=1000001000, 521=1000001001, 522=1000001010, 523=1000001011, 524=1000001100, 525=1000001101, 526=1000001110, 527=1000001111, 576=1001000000, 577=1001000001, 578=1001000010, 579=1001000011, 580=1001000100, 581=1001000101, 582=1001000110, 583=1001000111, 584=1001001000, 585=1001001001, 586=1001001010, 587=1001001011, 588=1001001100, 589=1001001101, 590=1001001110, 591=1001001111, 640=1010000000, 641=1010000001, 642=1010000010, 643=1010000011, 644=1010000100, 645=1010000101, 646=1010000110, 647=1010000111, 648=1010001000, 649=1010001001, 650=1010001010, 651=1010001011, 652=1010001100, 653=1010001101, 654=1010001110, 655=1010001111, 656=1010010000, 657=1010010001, 658=1010010010, 659=1010010011, 660=1010010100, 661=1010010101, 662=1010010110, 663=1010010111, 664=1010011000, 665=1010011001, 666=1010011010, 667=1010011011, 668=1010011100, 669=1010011101, 670=1010011110, 671=1010011111, 704=1011000000, 705=1011000001, 706=1011000010, 707=1011000011, 708=1011000100, 709=1011000101, 710=1011000110, 711=1011000111, 712=1011001000, 713=1011001001, 714=1011001010, 715=1011001011, 716=1011001100, 717=1011001101, 718=1011001110, 719=1011001111, 768=1100000000, 769=1100000001, 770=1100000010, 771=1100000011, 772=1100000100, 773=1100000101, 774=1100000110, 775=1100000111, 776=1100001000, 777=1100001001, 778=1100001010, 779=1100001011, 780=1100001100, 781=1100001101, 782=1100001110, 783=1100001111, 800=1100100000, 801=1100100001, 802=1100100010, 803=1100100011, 804=1100100100, 805=1100100101, 806=1100100110, 807=1100100111, 808=1100101000,

809=1100101001, 810=1100101010, 811=1100101011, 812=1100101100, 813=1100101101, 814=1100101110, 815=1100101111,
832=1101000000, 833=1101000001, 834=1101000010, 835=1101000011, 836=1101000100, 837=1101000101, 838=1101000110,
839=1101000111, 840=1101001000, 841=1101001001, 842=1101001010, 843=1101001011, 844=1101001100, 845=1101001101,
846=1101001110, 847=1101001111, 896=1110000000, 897=1110000001, 898=1110000010, 899=1110000011, 900=1110000100,
901=1110000101, 902=1110000110, 903=1110000111, 904=1110001000, 905=1110001001, 906=1110001010, 907=1110001011,
908=1110001100, 909=1110001101, 910=1110001110, 911=1110001111, 912=1110010000, 913=1110010001, 914=1110010010,
915=1110010011, 916=1110010100, 917=1110010101, 918=1110010110, 919=1110010111, 920=1110011000, 921=1110011001,
922=1110011010, 923=1110011011, 924=1110011100, 925=1110011101, 926=1110011110, 927=1110011111, 928=1110100000,
929=1110100001, 930=1110100010, 931=1110100011, 932=1110100100, 933=1110100101, 934=1110100110, 935=1110100111,
936=1110101000, 937=1110101001, 938=1110101010, 939=1110101011, 940=1110101100, 941=1110101101, 942=1110101110,
943=1110101111, 960=1111000000, 961=1111000001, 962=1111000010, 963=1111000011, 964=1111000100, 965=1111000101,
966=1111000110, 967=1111000111, 968=1111001000, 969=1111001001, 970=1111001010, 971=1111001011, 972=1111001100,
973=1111001101, 974=1111001110, 975=1111001111.

Theorem VI.4.b.1: There exist 200 classes of functions of three-valued set logic under compositions with [B] functions.▪

VI.5. Enumeration of B-bases of two-valued and three-valued set logic

A B-complete set F in P_k is called a B-base of P_k if no proper subset of F is B-complete in P_k. The rank of a B-base is the number of its elements. A non [B] function f ($f \in P_k \setminus B$) is a B-Sheffer for P_k if $\{f\}$ is a B-base (of rank 1) of P_k. In other words, f is a B-Sheffer function for P_k if $\{f\} \cup B$ is B-complete in P_k.

VI.5.a. Enumeration of B-bases in P_4

Lemma VI.5.a.1: The number of n-place B-Sheffer functions of two-valued set logic is

$$|P_4(n)| - 2|E_1(n)| - |C_0(n)| + |BF(n)| + 2|E_1(n) \cap C_0(n)| - |E_1(n) \cap E_2(n) \cap C_0(n)|.$$

Proof: According to the principle of inclusion and exclusion, the number of n-place B-Sheffer functions in P_4 is $|P_4(n) \setminus (E_1(n) \cup E_2(n) \cup C_0(n))| = |P_4(n)| - |E_1(n)| - |E_2(n)| - |C_0(n)| + |E_1(n) \cap E_2(n)| + |E_1(n) \cap C_0(n)| + |E_2(n) \cap C_0(n)| - |E_1(n) \cap E_2(n) \cap C_0(n)|$. Because of symmetry, $|E_1(n)| = |E_2(n)|$ and $|E_1(n) \cap C_0(n)| = |E_2(n) \cap C_0(n)|$. Also from [S3] $E_1(n) \cap E_2(n) = BF(n)$. This completes the proof.▪

Lemma VI.5.a.2: $|E_1(n) \cap C_0(n)| = 2^{4^n}$.

Proof: Let f be a n-place function of P_k such that f preserves both relations ε_1 and γ_0. Partition the set L_4^n into n-dimensional blocks in such a way that two points $x = (x_1, x_2, ..., x_n)$ and y

$= (y_1, y_2, ..., y_n)$, $x_j, y_j \in L_4$, belong to the same block iff for every $j = 1, 2, ..., n$, both x_j and y_j belong to the same equivalence class of ε_1. There are 2^n points in γ_0^n. Each block of the partition contains 2^n points, and then γ_0^n intersects each block in exactly one point we call an intersect-point. Consider a block and its intersect-point x. As $f \in C_0$, the value f(x) in the intersect-point x is either 0 or 3. Also as $f \in E_1$, the values f(y) in the other points of the block are all from the same equivalence class of ε_1 (either {0, 1} or {2, 3}) containing f(x). Therefore each point chooses between two values. Since there are 4^n points, then it follows that the number of functions preserving both relations ε_1 and γ_0 is 2^{4^n}.▪

Theorem VI.5.a.3: The number of n-place B-Sheffer functions of two-valued set logic is

$$4^{4^n} - 2^{4^n} + 2^n + 1 - 2^{2^n}.4^{4^n} - 2^n + 2^{2^n + 1} + 2^{4^n} + 1 - 2^{2^n}.$$

Proof: Follows from Theorems VI.3.1 and VI.3.11, Lemmas VI.3.8, VI.5.a.1 and VI.5.a.2.▪

The ratio of n-place B-Sheffer functions over all n-place two-valued set logic functions approaches 1 when $n \rightarrow +\infty$.

The program for classification and enumeration found all the classes of B-bases of rank 2 and 3 in P_4. There are six classes of B-bases of rank 2 (namely (7 6), (7 4), (7 2), (6 4), (6 3), (4 5)) and one class of B-bases of rank 3 (namely (5 3 2)) - the numbers here are the ordinal numbers of the binary characteristic vectors of the classes as they appears in section VI.4.a.

VI.5.b. Enumeration of B-bases in P_8

In P_8, it becomes hard to apply the principle of inclusion and exclusion to find the number of three-valued B-Sheffer functions. This is due to the large number of intersections of B-maximal sets: there are ten B-maximal sets in P_8 and then 1013 intersections. However, we can estimate the number of B-Sheffer functions by an interval, i.e. find a lower bound and an upper bound. To compute the upper bound and the lower bound we obtain the number of functions which are in no B-maximal sets defined from central relations on L_8. Next, we consider the number of functions which are in no B-maximal sets defined from equivalence relations on L_8, this number is given in [D1] as the number of n-place C-Sheffer functions of three-valued set logic (as we have explained, a C-maximal set from [D1] is also a B-maximal set in our context). The upper bound and the lower bound can then be determined using these two quantities. In the sequel of this section, we denote C(n) as the set of n-place functions preserving central relations on L_8

(i.e. $\cup C_j(n)$, j = 0, 1, 2, 4) and E(n) as the set of n-place functions preserving equivalence relations on L_8 (i.e. $\cup E_i(n)$, i = 1, 2, 3, 4, 5, 6). Also we let $\overline{C}(n)$ ($\overline{E}(n)$) be the set of set logic functions not preserving any central relations (equivalence relations, resp.).

Theorem VI.5.b.1: [D1] The number of n-place C-Sheffer functions of three-valued set logic is

$$|\overline{E}(n)| = 8^{8^n} - 3\cdot 2^{8^n}\cdot 4^{4^n} - 3\cdot 4^{8^n}\cdot 2^{2^n} + 6\cdot 2^{2^n}\cdot 2^{4^n}\cdot 2^{8^n} + 3\cdot 4^{4^n}\cdot 2^{2^n} - 6\cdot 2^{4^n}\cdot 2^{2^{n+1}} + 2\cdot 8^{2^n}.$$

Lemma VI.5.b.2: The number of n-place functions of three-valued set logic which are in no B-maximal sets defined from central relations on L_8 is

$$|\overline{C}(n)| = |P_8(n)| - |C_0(n)| - 3|C_1(n)| + 3|C_0(n) \cap C_1(n)|.$$

Proof: According to the principle of inclusion and exclusion, $|\overline{C}(n)| = |P_8(n) \setminus (C_0(n) \cup C_1(n) \cup C_2(n) \cup C_4(n))| = |P_8(n)| - |C_0(n)| - |C_1(n)| - |C_2(n)| - |C_4(n)| + |C_0(n) \cap C_1(n)| + |C_0(n) \cap C_2(n)| + |C_0(n) \cap C_4(n)| + |C_1(n) \cap C_2(n)| + |C_1(n) \cap C_4(n)| + |C_2(n) \cap C_4(n)| - |C_0(n) \cap C_1(n) \cap C_2(n)| - |C_0(n) \cap C_1(n) \cap C_4(n)| - |C_0(n) \cap C_2(n) \cap C_4(n)| - |C_1(n) \cap C_2(n) \cap C_4(n)| + |C_0(n) \cap C_1(n) \cap C_2(n) \cap C_4(n)|$. By symmetry $|C_1(n)| = |C_2(n)| = |C_4(n)|$, $|C_0(n) \cap C_1(n)| = |C_0(n) \cap C_2(n)| = |C_0(n) \cap C_4(n)|$, $|C_1(n) \cap C_2(n)| = |C_1(n) \cap C_4(n)| = |C_2(n) \cap C_4(n)|$, and $|C_0(n) \cap C_1(n) \cap C_2(n)| = |C_0(n) \cap C_1(n) \cap C_4(n)| = |C_0(n) \cap C_2(n) \cap C_4(n)|$. We obtain $|\overline{C}(n)| = |P_8(n)| - |C_0(n)| - 3|C_1(n)| + 3|C_0(n) \cap C_1(n)| + 3|C_1(n) \cap C_2(n)| - 3|C_0(n) \cap C_1(n) \cap C_2(n)| - |C_1(n) \cap C_2(n) \cap C_4(n)| + |C_0(n) \cap C_1(n) \cap C_2(n) \cap C_4(n)|$. From $A \cap B \subseteq C$ it follows that $A \cap B \cap C = A \cap B$. This is applied to Lemmas VI.3.6 and VI.3.7 to give $|C_0(n) \cap C_1(n) \cap C_2(n)| = |C_1(n) \cap C_2(n)|$ and $|C_0(n) \cap C_1(n) \cap C_2(n) \cap C_4(n)| = |C_1(n) \cap C_2(n) \cap C_4(n)|$. Finally we get $|\overline{C}(n)| = |P_8(n)| - |C_0(n)| - 3|C_1(n)| + 3|C_0(n) \cap C_1(n)|$.■

Lemma VI.5.b.3: $|C_0(n) \cap C_1(n)| = 2^{2^n}\cdot 4^{4^n} - 2^n\cdot 8^{8^n} - 4^n$.

Proof: We have $\gamma_0 = \{0, 7\} \subset \gamma_1 = \{0, 1, 6, 7\} \subset L_8$. Let f be a n-place function of P_8 such that f preserves both relations γ_0 and γ_1. Partition the set L_8^n into three distinct subsets $S_0 = \gamma_0^n$, $S_1 = \gamma_1^n \setminus \gamma_0^n$, and $S_2 = L_8^n \setminus (S_0 \cup S_1)$. As $f \in C_0(n)$ the values in the points of S_0 are all from γ_0; $|S_0| = 2^n$ implies that there are 2^{2^n} ways to fix the values in the points of S_0. As $f \in C_1(n)$ the values in the points of S_1 are from γ_1; $|S_1| = 4^n - 2^n$ implies that there are $4^{4^n - 2^n}$ ways to fix the values in the points of S_1. Finally, the values in the other points, that is in the points of S_2, are all from L_8; as $|S_2| = 8^n - 4^n$ then there are $8^{8^n - 4^n}$ ways to fix the values in

the points of S_2. It follows that the number of functions preserving both relations γ_0 and γ_1 is $2^{2^n}.4^{4^n} - 2^{2^n}.8^{8^n} - 4^{4^n}$.▪

Theorem VI.5.b.4: The number of n-place functions of three-valued set logic which are in no B-maximal sets defined from unary central relations on L_8 is

$$|\overline{C}(n)| = 8^{8^n} - 2^{2^n}.8^{8^n} - 2^{2^n} - 3.4^{4^n}.8^{8^n} - 4^{4^n} + 3.2^{2^n}.4^{4^n} - 2^{2^n}.8^{8^n} - 4^{4^n}.$$

Proof: Follows from Theorem VI.3.1, Lemmas VI.5.b.2 and VI.5.b.3.▪

Lemma 6.2.5: Let Sh(n) denote the set of n-place B-Sheffer functions in P_8, then

$$|\overline{E}(n)| - |C(n)| = |\overline{C}(n)| - |E(n)| < |Sh(n)| < |\overline{C}(n)| < |\overline{E}(n)|.$$

Proof: For n = 1 we have $|\overline{E}(1)| = 15894144$ and $|\overline{C}(1)| = 13369344$. So, as n grows, $|\overline{C}(n)| \leq |\overline{E}(n)|$ for $n \geq 0$ (for n = 0, $|\overline{C}(0)| = |\overline{E}(0)| = 0$). $\overline{C}(n)$ contains B-Sheffer functions, that is $Sh(n) \subset \overline{C}(n)$. $\overline{C}(n)$ contains also a set $U_1(n)$ of functions which preserve equivalence relations on L_8, thus $Sh(n) = \overline{C}(n) \setminus U_1(n)$. Next, $E(n) = \cup E_i(n)$ contains all functions which preserve at least one equivalence relation, it follows that $U_1(n) \subset E(n)$. Therefore we have $\overline{C}(n) \setminus E(n) \subset \overline{C}(n) \setminus U_1(n)$ and follows that $|\overline{C}(n)| - |E(n)| < |Sh(n)|$. Then we can take $|\overline{C}(n)| - |E(n)|$ as the lower bound of the number of B-Sheffer functions (the formula for |E(n)| appears in Theorem VI.5.b.1 after the term 8^{8^n}). Also, $\overline{E}(n)$ contains a set $U_2(n)$ of functions which preserve central relations on L_8, thus $Sh(n) = \overline{E}(n) \setminus U_2(n)$. Applying the same reasoning as above we obtain $|\overline{E}(n)| - |C(n)| < |Sh(n)|$ (the formula for |C(n)| appears in Theorem VI.5.b.4 after the term 8^{8^n}). Finally, we have $|\overline{E}(n)| - |C(n)| = |P_8(n)| - |E(n)| - |C(n)| = |P_8(n)| - |C(n)| - |E(n)| = |\overline{C}(n)| - |E(n)|$.▪

Obviously the ratios $\frac{|\overline{C}(n)|-|E(n)|}{|P_8(n)|}$ and $\frac{|\overline{C}(n)|}{|P_8(n)|}$ approach 1 when $n \to +\infty$, and therefore almost all three-valued set logic functions are B-Sheffer functions.

Using a computer program, we found that the rank of any B-base of P_8 is between 1 and 7.

Rank	Number of aggregates	Example of aggregates
1	1	(200)
2	4301	(199 198)
3	150973	(195 194 175)
4	230632	(193 191 190 140)
5	40166	(189 181 179 178 89)
6	1496	(177 165 104 92 55 12)
7	9	(134 126 119 91 78 15 4)

The numbers in the classes of B-bases are the ordinal numbers of the classes as they appear in section VI.4.b.

CHAPTER VII
OPEN PROBLEMS AND FURTHER STUDIES IN SET LOGIC ALGEBRA

In this thesis we have derived new results on classification and enumeration of C-bases in P_4 and P_8 and, on classification and enumeration of functions and B-bases in P_4 and P_8. Here we present some open problems.

1. Let $B_k(n)$ be the set of all n-place [B] functions in P_k. Determine $|B_k(n)|$, i.e. the exact number of such functions in P_k. We found only an upper bound in Lemma VI.3.10.

2. An interesting question is to determine all clones in set logic containing all Boolean functions ([D2,S3] find only maximal clones). Each C-clones and B-clones is the intersection of some maximal sets.

3. There are a number of remaining questions concerning S-completeness criteria in set logic if certain functions are considered cheap. Assuming first that S is any non empty subset of {$\cup$, $\cap$, $^-$, constants} we are going to determine the S-completeness criteria in set logic for subsets S other than C and B, the most interesting cases being S = {$\cup$, $\cap$}, S = {$^-$, constants} and S = {$^-$}. Second, S-completeness criteria in set logic for any given set S not necessarily Boolean is left as open problem, for example, when S = {$^-$, min(x, y)}.

4. Listing all classes of functions and classification and enumeration of C-bases and B-bases, C-Sheffer and B-Sheffer functions for r-valued set logic when $r \geq 4$ remains also an open problem for further study.

5. Recent developments in universal algebra have stimulated an interest in power structures (power algebras, in particular) [B2], that is, in structures defined on power domains where operations result by extending the usual operations on the base domain. It is interesting to explore the possibility of transferring (S-)completeness results from operations on sets to similar operations on power domains. Results from this area should be of interest for engineers who use functions defined on power sets to model properties of carrier computing systems.

6. There are open problems on approximation theory in set logic. In order to design the most economical hybrid circuits (that is circuits that combine set-valued biological or optical components -carrier computing devices- with standard binary electronic components) it is

necessary to extend the previous classification of one-place set logic functions based on the number of maximal interpolation classes [R1] to n-place functions, and to characterize almost-Boolean set-valued switching functions, that is, functions that have a small number of Boolean interpolation classes. It is interesting to investigate the metric properties of collections (classes) of set logic functions, i.e. properties that can be expressed by using suitable metrics. Special attention should be paid to the possibility of metric characterizations of maximal clones of (S-)complete sets of set logic functions.

7. It is also important to design fast (parallel) algorithms for the search of S-maximal sets in P_k, for the classification and enumeration of functions and S-bases of set logic, and for cataloging set logic functions based on their Boolean interpolation properties. Such algorithms are particularly necessary for $r \geq 4$ or $n \geq 2$.

8. The study of the complexity of set logic functions asks the question of how to construct them. The complexity of a function is defined as the least number of non Boolean components needed by a set logic circuit that computes the function (see section IV.6.a). The problem is then to find a S-complete set such that the set logic circuit of the function have a minimum number of non Boolean components.

APPENDIX 1
A PASCAL PROGRAM FOR CLASSIFICATION AND ENUMERATION OF SET LOGIC FUNCTIONS

A.1.1. Source codes

```
PROGRAM ClassesOfFunctions;
{This program classifies and enumerates set logic function under compositions with C or B functions}
{We use the algorithms we described in section V.2.}

uses crt;

const
   nbMaxClas = 1024;                    {maximum number of classes}
   nbMaxSets = 10;                      {maximum number of maximal sets (or relations)}
   Arity = 1;                           {maximum number of variables of a set logic function}
   nbPoints = 8;                        {maximum number of n-tuples taken out of k values (n = Arity, and k = 2^r)}
   nbEquiClas = 4;                      {maximum number of equivalence classes in an equivalence relation}

type
  variation = array[1..nbPoints] of integer;                          {array representation for a set logic function}
  stringclas = string[nbMaxSets];                                     {string representation for a characteristic vector}
  ntuples = array[1..nbPoints] of array[1..Arity] of longint;         {matrix containing the set of n-tuples (or points)}
  relation = array[1..nbEquiClas] of array[1..nbEquiClas] of integer; {matrix representation for a relation}
  classe = record                                                     {structure for a class of set logic functions}
             decimal : longint;                                       {decimal equivalent of its binary characteristic vector}
             vector  : stringclas;                                    {its binary characteristic vector}
             exFunct : variation;                                     {an example set logic function of the class}
             nbFunct : longint;                                       {its number of set logic functions}
           end;
  maxset = record                                                     {structure for a relation}
             nbEquiv : integer;                                       {its number of equivalence classes}
             nbElem  : array[1..nbEquiClas] of integer;               {the number of elements in each equivalence class}
             relat   : relation;                                      {the relation}
             nbFunct : longint;                                       {the number of set logic functions preserving it}
           end;
  tabmaxsets = array[1..nbMaxSets] of maxset;                         {array containing the set of relations}
  tabvectors = array[1..nbMaxClas] of classe;                         {array containing the set of classes of set logic functions}

var
  t, i : integer;
  c, s, r, k, n, kn, m, nbC, HalfCardinal, nf: longint;
  funqtion : variation;
  InpClasses, OutClasses : text;
  maxsets : tabmaxsets;
  vectors : tabvectors;
  binary : stringclas;
  points : ntuples;
  funq : string;

PROCEDURE ReadMaxSets(var maxsets : tabmaxsets);
{Reads the relations from a file and stores them into the maxsets array. See the input files format in section A.1.2.}

var
  s : longint;
  i, j : integer;

begin
   for s := 1 to m do                                                 {there are m equivalence relations to read}
   begin
      Read(InpClasses, maxsets[s].nbEquiv);                           {reads the number of equivalence classes of the relation}
      for i := 1 to maxsets[s].nbEquiv do                             {and the number of element of each equivalence class}
        Read(InpClasses, maxsets[s].nbElem[i]);
      Readln(InpClasses);
      for i := 1 to maxsets[s].nbEquiv do                             {reads the equivalence relation}
        for j := 1 to maxsets[s].nbElem[i] do
          Read(InpClasses, maxsets[s].relat[i][j]);
      maxsets[s].nbFunct := 0;
      Readln(InpClasses);
   end;
end;
```

```
PROCEDURE BuildPoints(var points : ntuples);
{Constructs the set of points using a well known base k conversion algorithm}

var
  p, i, decimal, quotient, base : longint;

begin
   base := k;
   for p := 1 to kn do                                   {there are k^n points}
   begin
      for i := 1 to n do
         points[p][i] := 0;
      i := n;
      decimal := p - 1;                                  {we convert the decimal number (p - 1) to its base k number}
      quotient := decimal;
      while (quotient > 0) and (i > 0) do
      begin
         quotient := decimal div base;
         points[p][i] := decimal mod base;               {the coordinates of a point are simply the digits of the base k number}
         decimal := quotient;
         i := i - 1;
      end;
   end;
end;

FUNCTION Power(int, pow : longint) : longint;
{Computes the function int^pow}

var
  i, tp : longint;

begin
   tp := 1;
   for i := 1 to pow do
      tp := tp * int;
   Power := tp;
end;

PROCEDURE ToString(funqtion : variation; var funq : string);
{Converts a set logic function into a string enclosed with parenthesis}

var
  s : longint;
  strfct : string;

begin
   funq := '(';
   for s := 1 to kn do
   begin
      Str(funqtion[s], strfct);
      funq := funq + strfct + ',';
   end;
   funq[Length(funq)] := ')';
end;

PROCEDURE Classify(funqtion : variation; var maxsets : tabmaxsets; var c : longint; var binary : stringclas);
{Finds the equivalence class of a given set logic function f and updates the size of all the maximal sets containing f}
{We use the (second) algorithm we describeb in section V.2.. So, for better understanding, read first the description}

var
  preserves, found, belongs : boolean;
  image, domain : array[1..nbPoints] of integer;
  img : array[1..nbEquiClas] of integer;
  j : integer;
  i, s, p, coord : longint;
  funq : string;
```

```
begin
   for s := 1 to m do                                          {check the inclusion in the m maximal sets}
   begin        {MAIN LOOP}
      binary[s] := '1';                                        {f is not included yet in the courant maximal set}
      for p := 1 to kn do                                      {looks for the image of k^n points}
      begin     {image loop}
         image[p] := 0;
         i := 1;
         found := False;
         while (found = False) and (i <= maxsets[s].nbEquiv) do   {looks for the image of the p-th point in the courant relation s}
         begin
            j := 1;
            while (found = False) and (j <= maxsets[s].nbElem[i]) do   {which equivalence class contains the map of the p-th point ?}
            begin
               found := funqtion[p] = maxsets[s].relat[i][j];
               j := j + 1;
            end;
            i := i + 1;
         end;
         if found = True then image[p] := i - 1;               {p-th point image := found class position, otherwise it is 0}
      end;      {image loop}

      for p := 1 to kn do                                      {looks for the domain of all points}
      begin     {domain loop}
         domain[p] := 0;
         i := 1;
         found := False;
         while (found = False) and (i <= maxsets[s].nbEquiv) do   {is the first coordinate of the p-th point in the relation s ?}
         begin
            j := 1;
            while (found = False) and (j <= maxsets[s].nbElem[i]) do   {in which equivalence class is it ?}
            begin
               found := points[p][1] = maxsets[s].relat[i][j];
               j := j + 1;
            end;
            i := i + 1;
         end;

         if found = True then                                  {if it is found then do the same for the other coordinates}
         begin  {domain if}
            i := i - 1;
            coord := 2;
            belongs := True;
            while (belongs = True) and (coord <= n) do        {n is the number of variables of f}
            begin
               j := 1;
               found := False;
               while (found = False) and (j <= maxsets[s].nbElem[i]) do
               begin
                  found := points[p][coord] = maxsets[s].relat[i][j];
                  j := j + 1;
               end;
               if found = False then belongs := False;
               coord := coord + 1;
            end;

            if belongs = True then                            {all p-th point coordinates are in the same equivalence class}
            begin
               domain[p] := i;                                {p-th point domain := found class position, otherwise it is 0}
               img[domain[p]] := image[p];
            end;
         end;   {domain if}
      end;      {domain loop}

      p := 1;
      preserves := True;
```

```
      while (preserves = True) and (p <= kn) do                    {f preserves the s-th relation if each non null domain maps to}
      begin                                                        {exactly one image}
         if domain[p] <> 0 then
           preserves := (image[p] <> 0) and (image[p] = img[domain[p]]);
         p := p + 1;
      end;
      if preserves = True then                                     {if f preserves the s-th relation, then...}
      begin
         binary[s] := '0';                                         {...the s-th bit of the characteristic vector is set to 0}
         maxsets[s].nbFunct := maxsets[s].nbFunct + 2;
      end;
   end;          {MAIN LOOP}

   c := 0;
   i := m - 1;
   for s := 1 to m do                                              {we compute the decimal equivalent of the binary vector}
   begin
      if binary[s] = '1' then
      begin
         c := c + Power(2, i);
      end;
      i := i - 1;
   end;
   c := c + 1;

   if ((vectors[c].nbFunct + 2) mod c) = 0 then                    {this is needed only for testing and checking}
   begin
      ToString(funqtion, funq);
      Writeln(c:4, ' ', binary:m, ' ', funq:Length(funq), ' ', (vectors[c].nbFunct+2):8, ' ', nf:8);
   end;
end;

(****MAIN PROGRAM****)

BEGIN
   Clrscr;
   Assign(InpClasses, 'classes.inp');
   Reset(InpClasses);
   Read(InpClasses, r);                                            {reads the number of carrier elements}
   k := Power(2, r);                                               {number of logic values: k = 2^r}
   Read(InpClasses, n);                                            {reads the number of variables of the set logic functions}
   kn := Power(k, n);                                              {number of points: kn = k^n}
   HalfCardinal := Power(k, kn) div 2;                             {number of set logic functions to classify: HalfCardinal = k^kn / 2}
   Readln(InpClasses, m);                                          {reads the number of equivalence relations (or maximal sets)}
   nbC := Power(2, m);                                             {number of possibles classes of set logic functions: nbC = 2^m}
   ReadMaxSets(maxsets);                                           {reads the relations}
   Close(InpClasses);
   BuildPoints(points);                                            {constructs the set of points (or n-tuples)}

   for c := 1 to nbMaxClas do                                      {all initialisations go here}
   begin
      vectors[c].decimal := -1;
      vectors[c].vector := '';
      for i := 1 to kn do
         vectors[c].exFunct[i] := -1;
      vectors[c].nbFunct := 0;
   end;
   binary := '';
   for s := 1 to m do
      binary := binary + '1';
   for i := 1 to kn do
      funqtion[i] := 0;
   nf := 0;                                                        {will contains the total number of set logic functions}
```

```
repeat                                          {we implement the variation algorithm. So, read section V.2.}
    nf := nf + 1;
    Classify(funqtion, maxsets, c, binary);     {we classify a set logic function}
    vectors[c].decimal := c - 1;                {then we update the array of classes from here}
    vectors[c].vector := binary;
    for i := 1 to kn do
      vectors[c].exFunct[i] := funqtion[i];
    vectors[c].nbFunct := vectors[c].nbFunct + 2;
    t := kn;                                    {from here we determine the next set logic function to classify}
    while funqtion[t] = k - 1 do
        t := t - 1;
    funqtion[t] := funqtion[t] + 1;
    for i := t + 1 to kn do
      funqtion[i] := 0;
until nf = HalfCardinal;                        {we classify only half of the set of set logic functions}

Assign(OutClasses, 'classes.out');              {then from here we output the results of the classification}
Rewrite(OutClasses);
i := 1;
nf := 0;
for c := 1 to nbC do
begin
   if vectors[c].nbFunct > 0 then
   begin
      ToString(vectors[c].exFunct, funq);
      Write(OutClasses, i:3, ' ', vectors[c].decimal:4, ' ');
      Write(OutClasses, vectors[c].vector:m, ' ', funq:Length(funq));
      Writeln(OutClasses, ' ', vectors[c].nbFunct:8);
      nf := nf + vectors[c].nbFunct;
      i := i + 1;
   end;
end;
Writeln(OutClasses);
Writeln(OutClasses, 'Total number of ', n, '-place ', r, '-valued set logic functions is : ', nf:8);
for s := 1 to m do
begin
   Write(OutClasses, 'The maximal set ', s:2, ' contains ', maxsets[s].nbFunct:8, ' ', n, '-place ', r);
   Writeln(OutClasses, '-valued set logic functions');
end;
Close(OutClasses);
END.
```

A.1.2. Input/Output files formats

Input

The first line of one input file contains three positive integers in the following order: r, n and m. Where r is the number of information carriers, n is the arity of the set logic functions, and m is the number of S-maximal sets. (We obtain results only for S = C and S = B).

Each of the m following groups of two lines represents a relation. The first line contains in this order: the number of equivalence classes of the relation and for each class its number of elements. The second line contains the equivalence relation. (For better understanding, see the input file for the case S = C, r = 2 and n = 1, below).

Output

The m + 1 last lines in one output file give information on the total number of set logic functions and the size of each S-maximal set. Any other line corresponds to a non empty class of set logic functions. In such line, we show in the following order: the ordinal and decimal numbers of the class, its characteristic vector, a one-place example function and the size of the class.

A.1.3. Input/Output files for S = C, r = 2 and n = 1

Input

```
2          1          2
2          2          2
           01         23
2          2          2
           02         13
```

Output

```
1   0 00 (1,1,3,3)    16
2   1 01 (1,1,3,2)    48
3   2 10 (1,3,3,3)    48
4   3 11 (1,3,3,2)   144

Total number of 1-place 2-valued set logic functions is :    256
The maximal set  1 contains     64 1-place 2-valued set logic functions
The maximal set  2 contains     64 1-place 2-valued set logic functions
```

A.1.4. Input/Output files for S = C, r = 3 and n = 1

Input

```
3          1          6
4          2          2          2          2
           01         23         45         67
4          2          2          2          2
           02         13         46         57
4          2          2          2          2
           04         15         26         37
2          4          4
           0246       1357
2          4          4
           0145       2367
2          4          4
           0123       4567
```

Output

```
 1   0 000000 (3,3,3,3,7,7,7,7)      64
 2  10 000010 (3,3,3,3,7,7,5,5)     192
 3  12 001100 (3,3,3,3,7,6,7,6)     192
 4  14 001110 (3,3,3,3,7,6,5,4)     576
 5  17 010000 (3,3,7,7,7,7,7,7)     192
 6  20 010100 (3,3,3,2,7,7,7,6)     192
 7  21 010101 (3,3,7,6,7,7,7,6)     576
 8  27 011011 (3,3,7,7,7,7,5,5)     576
 9  28 011100 (3,3,3,3,7,7,7,6)    3648
10  29 011101 (3,3,7,7,7,7,7,6)   11520
11  30 011110 (3,3,3,3,7,7,5,4)   11520
12  31 011111 (3,3,7,7,7,7,5,4)   36288
13  33 100001 (3,7,3,7,7,7,7,7)     192
14  34 100010 (3,3,3,1,7,7,7,5)     192
15  35 100011 (3,7,3,5,7,7,7,5)     576
16  42 101010 (3,3,3,3,7,7,7,5)    3648
17  43 101011 (3,7,3,7,7,7,7,5)   11520
18  45 101101 (3,7,3,7,7,6,7,6)     576
19  46 101110 (3,3,3,3,7,6,7,4)   11520
20  47 101111 (3,7,3,7,7,6,7,4)   36288
21  49 110001 (3,7,7,7,7,7,7,7)    3648
22  51 110011 (3,7,7,5,7,7,7,5)   11520
23  53 110101 (3,7,7,6,7,7,7,6)   11520
24  54 110110 (3,3,3,0,7,7,7,4)     576
25  55 110111 (3,7,7,4,7,7,7,4)   36288
26  59 111011 (3,7,7,7,7,7,7,5)  229824
27  61 111101 (3,7,7,7,7,7,7,6)  229824
28  62 111110 (3,3,3,3,7,7,7,4)  229824
29  63 111111 (3,7,7,7,7,7,7,4) 15894144

Total number of 1-place 3-valued set logic functions is : 16777216
The maximal set  1 contains   65536 1-place 3-valued set logic functions
The maximal set  2 contains   65536 1-place 3-valued set logic functions
The maximal set  3 contains   65536 1-place 3-valued set logic functions
The maximal set  4 contains  262144 1-place 3-valued set logic functions
The maximal set  5 contains  262144 1-place 3-valued set logic functions
The maximal set  6 contains  262144 1-place 3-valued set logic functions
```

A.1.5. Input/Output files for S = B, r = 2 and n = 1

Input

```
2        1        3
2        2        2
         01       23
2        2        2
         02       13
1        2
         03
```

Output

```
 1  0 000 (0,1,2,3)    4
 2  1 001 (1,1,3,3)   12
 3  2 010 (0,1,3,3)   12
 4  3 011 (1,1,3,2)   36
 5  4 100 (0,3,2,3)   12
 6  5 101 (1,3,3,3)   36
 7  6 110 (0,3,3,3)   36
 8  7 111 (1,3,3,2)  108

Total number of 1-place 2-valued set logic functions is :   256
The maximal set  1 contains    64 1-place 2-valued set logic functions
The maximal set  2 contains    64 1-place 2-valued set logic functions
The maximal set  3 contains    64 1-place 2-valued set logic functions
```

A.1.6. Input/Output files for S = B, r = 3 and n = 1

Input

```
3        1        10
4        2        2        2        2
         01       23       45       67
4        2        2        2        2
         02       13       46       57
4        2        2        2        2
         04       15       26       37
2        4        4
         0246     1357
2        4        4
         0145     2367
2        4        4
         0123     4567
1        2
         07
1        4
         0167
1        4
         0257
1        4
         0347
```

Output

```
 1   0 0000000000 (0,1,2,3,4,5,6,7)      4
 2  11 0000001011 (1,1,3,3,5,5,7,7)     12
 3  13 0000001101 (2,3,2,3,6,7,6,7)     12
 4  14 0000001110 (3,3,3,3,7,7,7,7)     12
 5  15 0000001111 (3,3,1,1,7,7,5,5)     24
 6 161 0010010001 (0,1,2,3,6,7,6,7)     12
 7 171 0010101011 (1,1,3,3,7,7,7,7)     36
 8 173 0010101101 (2,3,2,3,6,7,4,5)     36
 9 175 0010101111 (3,3,3,3,7,7,5,5)    108
10 193 0011000001 (0,1,2,3,5,5,7,7)     12
11 203 0011001011 (1,1,3,3,5,4,7,6)     36
12 205 0011001101 (2,3,2,3,7,7,7,7)     36
13 207 0011001111 (3,3,3,3,7,6,7,6)    108
14 224 0011100000 (0,1,2,3,7,7,7,7)     12
15 225 0011100001 (0,1,2,3,2,2,0,0)     24
16 235 0011101011 (1,1,3,3,7,6,7,6)    108
17 237 0011101101 (2,3,2,3,7,7,5,5)    108
18 238 0011101110 (3,3,3,3,7,6,5,4)     36
19 239 0011101111 (3,3,3,3,6,7,4,5)    288
20 274 0100010010 (0,1,6,7,4,5,6,7)     12
21 283 0100011011 (1,1,7,7,5,5,7,7)     36
22 286 0100011110 (3,3,7,7,7,7,7,7)     36
23 287 0100011111 (3,3,5,5,7,7,5,5)    108
24 322 0101000010 (0,1,3,3,4,5,7,7)     12
25 331 0101001011 (1,1,3,2,5,5,7,6)     36
26 334 0101001110 (3,3,2,3,7,7,6,7)     36
27 335 0101001111 (3,3,3,2,7,7,7,6)    108
28 336 0101010000 (0,1,7,7,4,5,7,7)     12
29 338 0101010010 (0,1,7,7,0,1,7,7)     24
30 347 0101011011 (1,1,7,6,5,5,7,6)    108
31 349 0101011101 (2,3,7,7,6,7,7,7)     36
32 350 0101011110 (3,3,6,7,7,7,6,7)    108
33 351 0101011111 (3,3,7,6,7,7,7,6)    288
34 435 0110110011 (0,1,6,7,6,7,6,7)     36
35 443 0110111011 (1,1,7,7,7,7,7,7)    108
36 447 0110111111 (3,3,7,7,7,7,5,5)    432
37 448 0111000000 (0,1,2,3,4,5,7,7)     12
38 449 0111000001 (0,1,2,3,5,5,6,7)     36
39 450 0111000010 (0,1,3,3,4,5,6,7)     36
40 451 0111000011 (0,1,3,3,5,5,7,7)    144
41 459 0111001011 (1,1,3,3,5,5,7,6)    684
42 461 0111001101 (2,3,2,3,7,7,6,7)    144
43 462 0111001110 (3,3,3,3,7,7,6,7)    144
44 463 0111001111 (3,3,3,3,7,7,7,6)   2448
45 464 0111010000 (0,1,7,7,4,5,6,7)     36
46 465 0111010001 (0,1,7,7,5,5,7,7)    144
47 466 0111010010 (0,1,7,7,4,4,7,7)    108
48 467 0111010011 (0,1,7,7,5,4,7,7)    432
49 475 0111011011 (1,1,7,7,5,5,7,6)   2160
50 477 0111011101 (2,3,7,7,7,7,7,7)    540
51 478 0111011110 (3,3,7,7,7,7,6,7)    432
52 479 0111011111 (3,3,7,7,7,7,7,6)   7668
53 480 0111100000 (0,1,2,3,7,7,6,7)     36
54 481 0111100001 (0,1,2,3,6,7,7,7)    108
55 482 0111100010 (0,1,3,3,7,7,7,7)    144
56 483 0111100011 (0,1,3,3,6,7,7,7)    432
57 491 0111101011 (1,1,3,3,7,7,7,6)   2160
58 493 0111101101 (2,3,2,3,7,7,4,5)    432
59 494 0111101110 (3,3,3,3,7,7,5,4)    540
60 495 0111101111 (3,3,3,3,7,7,4,5)   7668
61 496 0111110000 (0,1,7,7,7,7,7,7)    144
62 497 0111110001 (0,1,7,7,6,7,7,7)    432
63 498 0111110010 (0,1,7,7,7,6,7,7)    432
64 499 0111110011 (0,1,7,7,6,6,7,7)   1260
65 507 0111111011 (1,1,7,7,7,7,7,6)   6804
66 509 0111111101 (2,3,7,7,7,7,5,5)   1728
67 510 0111111110 (3,3,7,7,7,7,5,4)   1728
68 511 0111111111 (3,3,7,7,7,7,4,5)  23760
69 532 1000010100 (0,5,2,7,4,5,6,7)     12
70 541 1000011101 (2,7,2,7,6,7,6,7)     36
71 542 1000011110 (3,7,3,7,7,7,7,7)     36
72 543 1000011111 (3,7,1,5,7,7,5,5)    108
73 548 1000100100 (0,3,2,3,4,7,6,7)     12
74 557 1000101101 (2,3,2,1,6,7,6,5)     36
75 558 1000101110 (3,3,1,3,7,7,5,7)     36
76 559 1000101111 (3,3,3,1,7,7,7,5)    108
77 560 1000110000 (0,7,2,7,4,7,6,7)     12
78 564 1000110100 (0,7,2,7,0,7,2,7)     24
79 571 1000111011 (1,7,3,7,5,7,7,7)     36
80 573 1000111101 (2,7,2,5,6,7,6,5)    108
81 574 1000111110 (3,7,1,7,7,7,5,7)    108
82 575 1000111111 (3,7,3,5,7,7,7,5)    288
83 672 1010100000 (0,1,2,3,4,7,6,7)     12
84 673 1010100001 (0,1,2,3,6,5,6,7)     36
85 676 1010100100 (0,3,2,3,4,7,4,7)     36
```

86	677	1010100101	(0,3,2,3,6,7,6,7)	144
87	683	1010101011	(1,1,3,3,7,5,7,7)	144
88	685	1010101101	(2,3,2,3,6,7,6,5)	684
89	686	1010101110	(3,3,3,3,7,7,5,7)	144
90	687	1010101111	(3,3,3,3,7,7,7,5)	2448
91	688	1010110000	(0,7,2,7,4,5,6,7)	36
92	689	1010110001	(0,7,2,7,6,7,6,7)	144
93	692	1010110100	(0,7,2,7,4,7,4,7)	108
94	693	1010110101	(0,7,2,7,6,7,4,7)	432
95	699	1010111011	(1,7,3,7,7,7,7,7)	540
96	701	1010111101	(2,7,2,7,6,7,6,5)	2160
97	702	1010111110	(3,7,3,7,7,7,5,7)	432
98	703	1010111111	(3,7,3,7,7,7,7,5)	7668
99	725	1011010101	(0,5,2,7,5,5,7,7)	36
100	733	1011011101	(2,7,2,7,7,7,7,7)	108
101	735	1011011111	(3,7,3,7,7,6,7,6)	432
102	736	1011100000	(0,1,2,3,7,5,7,7)	36
103	737	1011100001	(0,1,2,3,5,7,7,7)	108
104	740	1011100100	(0,3,2,3,7,7,7,7)	144
105	741	1011100101	(0,3,2,3,5,7,7,7)	432
106	747	1011101011	(1,1,3,3,7,4,7,6)	432
107	749	1011101101	(2,3,2,3,7,7,7,5)	2160
108	750	1011101110	(3,3,3,3,7,6,7,4)	540
109	751	1011101111	(3,3,3,3,7,6,5,6)	7668
110	752	1011110000	(0,7,2,7,7,7,7,7)	144
111	753	1011110001	(0,7,2,7,6,2,6,0)	432
112	756	1011110100	(0,7,2,7,7,7,5,7)	432
113	757	1011110101	(0,7,2,7,6,2,4,0)	1260
114	763	1011111011	(1,7,3,7,7,6,7,6)	1728
115	765	1011111101	(2,7,2,7,7,7,7,5)	6804
116	766	1011111110	(3,7,3,7,7,6,7,4)	1728
117	767	1011111111	(3,7,3,7,7,6,5,6)	23760
118	784	1100010000	(0,1,2,7,4,5,6,7)	12
119	786	1100010010	(0,1,6,7,4,1,6,7)	36
120	788	1100010100	(0,5,2,7,4,5,2,7)	36
121	790	1100010110	(0,5,6,7,4,5,6,7)	144
122	795	1100011011	(1,1,7,7,5,1,7,7)	144
123	797	1100011101	(2,7,2,7,6,7,2,7)	144
124	798	1100011110	(3,7,7,7,7,7,7,7)	684
125	799	1100011111	(3,7,5,5,7,7,5,5)	2448
126	816	1100110000	(0,7,2,7,0,7,6,7)	36
127	818	1100110010	(0,7,6,7,4,7,6,7)	144
128	820	1100110100	(0,7,2,7,4,7,2,7)	108
129	822	1100110110	(0,7,6,7,4,7,2,7)	432
130	827	1100111011	(1,7,7,7,5,7,7,7)	540
131	829	1100111101	(2,7,2,5,6,7,2,5)	432
132	830	1100111110	(3,7,5,7,7,7,5,7)	2160
133	831	1100111111	(3,7,7,5,7,7,7,5)	7668
134	848	1101010000	(0,1,7,7,0,5,7,7)	36
135	850	1101010010	(0,1,7,7,4,1,7,7)	108
136	852	1101010100	(0,5,7,7,4,5,7,7)	144
137	854	1101010110	(0,5,7,7,4,1,7,7)	432
138	859	1101011011	(1,1,7,6,5,1,7,6)	432
139	861	1101011101	(2,7,7,7,6,7,7,7)	540
140	862	1101011110	(3,7,6,7,7,7,6,7)	2160
141	863	1101011111	(3,7,7,6,7,7,7,6)	7668
142	870	1101100110	(0,3,3,3,4,7,7,7)	36
143	878	1101101110	(3,3,3,0,7,7,7,4)	108
144	879	1101101111	(3,3,2,1,7,7,6,5)	432
145	880	1101110000	(0,7,7,7,4,7,7,7)	144
146	882	1101110010	(0,7,7,7,4,3,7,7)	432
147	884	1101110100	(0,7,7,7,4,7,3,7)	432
148	886	1101110110	(0,7,7,7,4,3,3,7)	1260
149	891	1101111011	(1,7,7,6,5,7,7,6)	1728
150	893	1101111101	(2,7,7,5,6,7,7,5)	1728
151	894	1101111110	(3,7,7,4,7,7,7,4)	6804
152	895	1101111111	(3,7,6,5,7,7,6,5)	23760
153	944	1110100000	(0,7,2,7,4,7,0,7)	144
154	945	1110100001	(0,7,2,7,6,7,0,7)	576
155	946	1110100010	(0,7,6,7,4,7,0,7)	576
156	947	1110100011	(0,7,6,7,6,7,6,7)	2268
157	948	1110100100	(0,7,2,7,4,5,2,7)	432
158	949	1110100101	(0,7,2,7,6,7,2,7)	1728
159	950	1110100110	(0,7,6,7,4,7,4,7)	1728
160	951	1110100111	(0,7,6,7,6,7,4,7)	6912
161	955	1110111011	(1,7,7,7,7,7,7,7)	10692
162	957	1110111101	(2,7,2,7,6,7,2,5)	8640
163	958	1110111110	(3,7,7,7,7,7,5,7)	8640
164	959	1110111111	(3,7,7,7,7,7,7,5)	187488
165	976	1111010000	(0,1,7,7,4,0,7,7)	144
166	977	1111010001	(0,1,7,7,5,0,7,7)	576
167	978	1111010010	(0,1,7,7,4,1,6,7)	432
168	979	1111010011	(0,1,7,7,5,1,7,7)	1728
169	980	1111010100	(0,5,7,7,4,5,6,7)	576
170	981	1111010101	(0,5,7,7,5,5,7,7)	2268
171	982	1111010110	(0,5,7,7,4,4,7,7)	1728
172	983	1111010111	(0,5,7,7,5,4,7,7)	6912
173	987	1111011011	(1,1,7,7,5,1,7,6)	8640
174	989	1111011101	(2,7,7,7,7,7,7,7)	10692
175	990	1111011110	(3,7,7,7,7,7,6,7)	8640
176	991	1111011111	(3,7,7,7,7,7,7,6)	187488
177	992	1111100000	(0,1,2,3,7,5,6,7)	144
178	993	1111100001	(0,1,2,3,6,5,7,7)	432
179	994	1111100010	(0,1,3,3,7,5,7,7)	576
180	995	1111100011	(0,1,3,3,6,5,7,7)	1728
181	996	1111100100	(0,3,2,3,7,7,6,7)	576
182	997	1111100101	(0,3,2,3,6,7,7,7)	1728
183	998	1111100110	(0,3,3,3,7,7,7,7)	2268
184	999	1111100111	(0,3,3,3,6,7,7,7)	6912
185	1003	1111101011	(1,1,3,3,7,5,7,6)	8640
186	1005	1111101101	(2,3,2,3,7,7,6,5)	8640
187	1006	1111101110	(3,3,3,3,7,7,7,4)	10692
188	1007	1111101111	(3,3,3,3,7,7,6,5)	187488
189	1008	1111110000	(0,7,7,7,7,7,7,7)	15228
190	1009	1111110001	(0,7,7,7,6,7,7,7)	46080
191	1010	1111110010	(0,7,7,7,7,6,7,7)	46080
192	1011	1111110011	(0,7,7,7,6,6,7,7)	139428
193	1012	1111110100	(0,7,7,7,7,5,7,7)	46080
194	1013	1111110101	(0,7,7,7,6,7,5,7)	139428
195	1014	1111110110	(0,7,7,7,7,6,5,7)	139428
196	1015	1111110111	(0,7,7,7,6,6,5,7)	421632
197	1019	1111111011	(1,7,7,7,7,7,7,6)	740448
198	1021	1111111101	(2,7,7,7,7,7,7,5)	740448
199	1022	1111111110	(3,7,7,7,7,7,7,4)	740448
200	1023	1111111111	(3,7,7,7,7,7,6,5)	12679416

Total number of 1-place 3-valued set logic functions is : 16777216
The maximal set 1 contains 65536 1-place 3-valued set logic functions
The maximal set 2 contains 65536 1-place 3-valued set logic functions
The maximal set 3 contains 65536 1-place 3-valued set logic functions
The maximal set 4 contains 262144 1-place 3-valued set logic functions
The maximal set 5 contains 262144 1-place 3-valued set logic functions
The maximal set 6 contains 262144 1-place 3-valued set logic functions
The maximal set 7 contains 1048576 1-place 3-valued set logic functions
The maximal set 8 contains 1048576 1-place 3-valued set logic functions
The maximal set 9 contains 1048576 1-place 3-valued set logic functions
The maximal set 10 contains 1048576 1-place 3-valued set logic functions

APPENDIX 2
A PASCAL PROGRAM FOR CLASSIFICATION AND ENUMERATION OF (S-)BASES (OF SET LOGIC)

A.2. Source codes

```
PROGRAM ClassesOfBases;
{This program classifies and enumerates (S-)bases}
{We use the algorithms we descrit ed in section V.3.}

uses crt;

const
   nbMaxSets = 10;                                   {maximum number of maximal sets}
   nbMaxClas = 200;                                  {maximum number of classes of functions}

type
   serie = array[1..nbMaxSets] of longint;           {array representation for a combination}
   pbase = ^lbase;                                   {pointer to a list of aggregates}
   lbase = record                                    {structure for an aggregate (or class of bases)}
             classe : string;                        {the aggregate}
             next   : pbase;                         {the next aggregate}
             product : longint;                      {the number of bases in the aggregate}
           end;
   aggregate = record                                {structure for a rank r}
                 nbBases : longint;                  {its number of aggregates (of rank r)}
                 bases   : pbase;                    {its list of aggregates (or rank r)}
               end;

var
  nbSets, nbClasses : longint;
  InpClasses, OutClasses : text;
  binVector : array[1..nbMaxClas] of string;         {the binary characteristic vectors}
  decVector : longint;
  vectors, nbOnes, sortVectors, ordVectors : array[1..nbMaxClas] of longint;   {decimal vectors and other informations}
  combination : serie;
  classeOfBases : array[1..nbMaxSets] of aggregate;
  last : array[1..nbMaxSets] of pbase;               {pointer to last aggregate for each rank}
  k, l, r, onesNb, prodbase : longint;
  pclasse : pbase;
  sbase, sb : string;
  bitwisor : array[1..nbMaxSets] of longint;         {keeps the bitwise or operations of some r-subsets}
  redbitor, total, summing : longint;                {redbitor: reduces the number of bitwise or's}
  sizes, sortSizes, ordSizes : array[1..nbMaxClas] of longint;   {the size of the characteristic binary vectors}

FUNCTION Power(int, pow : longint) : longint;
{Computes the function int^pow}

var
  i, tp : longint;

begin
   tp := 1;
   for i := 1 to pow do
     tp := tp * int;
   Power := tp;
end;

PROCEDURE BinToDec(binary : string; var decimal : longint);
{Converts a binary characteristic vector to its decimal equivalent}
{It counts also the number of unit (onesNb) in the binary vector}

var
  i, j : longint;
```

```
begin
   onesNb := 0;
   decimal := 0;
   i := nbSets - 1;
   for j := 1 to nbSets do
   begin
      if binary[j] = '1' then
      begin
         decimal := decimal + Power(2, i);
         onesNb := onesNb + 1;
      end;
      i := i - 1;
   end;
end;
```

FUNCTION IsRedundant(subset : serie) : boolean;
{Implements the *bitwise redundancy checks* algorithm we described in section V.3.c.}
{So, for better understanding, read first the description of the algorithm}

```
var
  i : longint;
  redundant : boolean;

begin
    IsRedundant := False;
    if r >= 2 then                                          {redundancy checks for a r-subset, r ≥ 2}
    begin          {FIRST BEGIN}
      bitwisor[r] := bitwisor[r - 1] or vectors[subset[r]];  {there is redundancy if T_r := T_r-1 ∨ r-th element of the...}
      if bitwisor[r] = bitwisor[r - 1] then                  {...r-subset and T_r = T_r-1, see section V.3.c.. T_s = bitwisor[s]}
        IsRedundant := True
      else
      begin        {second begin}
         redbitor := 0;                                      {will contains the bitwise or's from r-th element to 2nd one}
         i := r - 1;

         repeat
            redbitor := redbitor or vectors[subset[i + 1]];
            if i >= 2 then
            begin
               if bitwisor[i - 1] or redbitor = bitwisor[r] then   {there is redundancy if T_r = T_i-1 ∨ redbitor for i ≥ 2, ...}
               begin
                  redundant := True;
                  IsRedundant := True;
               end;
            end
            else
               if i = 1 then
                 if redbitor = bitwisor[r] then                    {...or T_r = redbitor for i = 1}
                 begin
                    redundant := True;
                    IsRedundant := True;
                 end;
             i := i - 1;
          until (redundant = True) or (i = 0);

       end;        {second begin}
    end            {FIRST BEGIN}
    else                                                    {redundancy check for a 1-subset}
    begin
       bitwisor[r] := vectors[subset[r]];                   {bitwisor[1] is the decimal vector element of the 1-subset}
       if (r = 1) and (vectors[subset[r]] = 0) then         {redundant only if the decimal vector is the null vector}
         IsRedundant := True;
    end;
  end;
```

PROCEDURE Extend(var subset : serie);
{Returns the next r-subset from the current row of the lexicographic generation. See section V.3.a.}

```
begin
   subset[r + 1] := subset[r] + 1;
   r := r + 1;
end;

PROCEDURE Reduce(var subset : serie);
{Returns the first r-subset from the next row of the lexicographic generation. See section V.3.a.}

begin
   r := r - 1;
   subset[r] := subset[r] + 1;
end;

PROCEDURE Cut(var subset : serie);
{Bypasses the lexicographic generation of the next r-subset and returns a r-subset from a lower row. See section V.3.a.}

begin
   if subset[r] < nbClasses then
      subset[r] := subset[r] + 1
   else
      Reduce(subset);
end;

(****MAIN PROGRAM****)

BEGIN
   Clrscr;
   Assign(InpClasses, 'bases.inp');
   Reset(InpClasses);
   Readln(InpClasses, nbSets, nbClasses);          {reads the number of maximal sets and classes of functions}
   for k := 1 to nbClasses do                      {reads all the binary characteristic vectors and their sizes}
   begin                                           {the vectors are already sorted in increasing order}
      Readln(InpClasses, binVector[k], sizes[k]);
      BinToDec(binVector[k], decVector);           {converts a binary vector to a decimal vector}
      vectors[k] := decVector;                     {we will deal with decimal vectors only, that's easier}
      nbOnes[k] := onesNb;                         {we store the number of units of a binary vector}
   end;
   Close(InpClasses);

   l := 0;
   for onesNb := nbSets downto 0 do                {sorts the vector according to their number...}
      for k := nbClasses downto 1 do               {...of units in non increasing order}
         if nbOnes[k] = onesNb then
         begin
            l := l + 1;
            sortVectors[l] := vectors[k];
            ordVectors[l] := k;                    {we store the original position of each vector}
            sortSizes[l] := sizes[k];              {we do the same for the sizes of the vectors}
            ordSizes[l] := k;
         end;
   for k := 1 to nbClasses do                      {new vectors and sizes arrays}
   begin
      vectors[k] := sortVectors[k];
      sizes[k] := sortSizes[k];
   end;

   for k := 1 to nbSets do                         {all initialisations go here}
   begin
      combination[k] := 0;
      bitwisor[k] := 0;
      classeOfBases[k].nbBases := 0;
      New(pclasse);
      last[k] := pclasse;
      last[k]^.classe := '';
      last[k]^.next := nil;
      classeOfBases[k].bases := last[k];
   end;
   r := 1;
   combination[1] := 1;
   bitwisor[1] := vectors[combination[1]];
```

```
repeat                                                          {lexicographic generation of r-subsets}
    if IsRedundant(combination) then
      Cut(combination)                                          {the current r-subset is redundant}
    else
      if bitwisor[r] <> Power(2, nbSets) - 1 then
        if combination[r] < nbClasses then                      {the current r-subset is addable}
          Extend(combination)
        else
          Reduce(combination)
      else                                                      {the r-subset is a (class of) bases}
      begin                                                     {then we update the rank r structure}
        classeOfBases[r].nbBases := classeOfBases[r].nbBases + 1;
        sbase := '()';
        prodbase := 1;
        for k := 1 to r do
        begin
          Str(ordVectors[combination[k]], sb);
          sb := sb + ' ';
          Insert(sb, sbase, Length(sbase));
          prodbase := prodbase * sizes[combination[k]];         {the size of the new aggregate}
        end;
        Delete(sbase, Length(sbase) - 1, 1);
        New(pclasse);                                           {from here we add the new aggregate}
        pclasse^.classe := '';
        pclasse^.product := 0;
        pclasse^.next := nil;
        last[r]^.classe := sbase;
        last[r]^.product := prodbase;
        last[r]^.next := pclasse;
        last[r] := pclasse;
        Cut(combination);                                       {...and proceed to the next r-subset}
      end;
until r = 0;

Assign(OutClasses, 'bases.out');                                {from here we output the results...}
Rewrite(OutClasses);                                            {...of classification and enumeration}
total := 0;
for k := 1 to nbSets do
begin
  Writeln(OutClasses, classeOfBases[k].nbBases, ' aggregates of rank ', k, ' :');
  pclasse := classeOfBases[k].bases;
  Write(OutClasses, '      ');
  summing := 0;
  for l := 1 to classeOfBases[k].nbBases do
  begin
    if l mod 5 <> 0 then
    begin
      write(OutClasses, pclasse^.classe, ' , ');
      summing := summing + pclasse^.product;
    end
    else
    begin
      Writeln(OutClasses, pclasse^.classe);
      Write(OutClasses, '      ');
      summing := summing + pclasse^.product;
    end;
    pclasse := pclasse^.next;
  end;
  Writeln(OutClasses, 'Number of bases of rank ', k, ' = ', summing);
  Writeln(OutClasses);
  Writeln(OutClasses);
  total := total + summing;
end;
Writeln(OutClasses, 'Total number of bases is ', total);
Close(OutClasses);
END.
```

A.2.2. Input/Output files formats

Input

The first line of one input file contains two positive integers in the following order: the number of S-maximal sets and the number of non empty classes of set logic functions. (We obtain results only for S = C and S = B).

Each of the following lines contains a class (characteristic vector) and its size. (For better understanding, see the input file for the case S = C, r = 2 and n = 1, below).

Output

See the output files below. The numbers into parenthesis are the ordinal number of the classes as they appear in the input files.

A.2.3. Input/Output files for S = C, r = 2 and n = 1

Input

```
2       4
00      16
01      48
10      48
11      144
```

Output

```
1 aggregates of rank 1 :
   (4)
   Number of bases of rank 1 = 144

1 aggregates of rank 2 :
   (3 2)
   Number of bases of rank 2 = 2304

Total number of bases is 2448
```

A.2.4. Input/Output files for S = C, r = 3 and n = 1

Input

```
6       29
000000  64
000010  192
000100  192
000110  576
001000  192
001010  192
001100  576
001110  576
010001  192
010010  192
010100  576
010110  576
011001  3648
011010  11520
011100  11520
011110  36288
100001  192
100010  192
100011  576
101010  3648
101011  11520
101101  576
101110  11520
101111  36288
110001  3648
110011  11520
110101  11520
110110  576
110111  36288
111011  229824
111101  229824
111110  229824
111111  15894144
```

Output

```
1 aggregates of rank 1 :
   (29)
   Number of bases of rank 1 = 15894144

120 aggregates of rank 2 :
   (28 27) , (28 26) , (28 25) , (28 20) , (28 12)
   (28 23) , (28 22) , (28 18) , (28 17) , (28 10)
   (28 8) , (28 21) , (28 15) , (28 7) , (28 13)
   (28 5) , (27 26) , (27 25) , (27 20) , (27 12)
   (27 24) , (27 22) , (27 19) , (27 17) , (27 11)
   (27 8) , (27 16) , (27 15) , (27 4) , (27 14)
   (27 2) , (26 25) , (26 20) , (26 12) , (26 24)
   (26 23) , (26 19) , (26 18) , (26 11) , (26 10)
   (26 9) , (26 7) , (26 4) , (26 6) , (26 3)
   (25 20) , (25 12) , (25 19) , (25 18) , (25 17)
   (25 11) , (25 10) , (25 8) , (25 16) , (25 9)
   (25 4) , (25 3) , (25 2) , (20 12) , (20 24)
   (20 23) , (20 22) , (20 11) , (20 10) , (20 8)
   (20 21) , (20 9) , (20 7) , (20 6) , (20 5)
   (12 24) , (12 23) , (12 22) , (12 19) , (12 18)
   (12 17) , (12 21) , (12 16) , (12 15) , (12 14)
   (12 13) , (24 18) , (24 17) , (24 10) , (24 8)
   (23 19) , (23 17) , (23 11) , (23 8) , (23 16)
   (23 4) , (23 2) , (22 19) , (22 18) , (22 11)
   (22 10) , (22 9) , (22 4) , (22 3) , (19 10)
   (19 8) , (19 21) , (19 7) , (19 5) , (18 11)
   (18 8) , (17 11) , (17 10) , (17 9) , (17 7)
   (17 6) , (11 21) , (11 15) , (11 13) , (10 16)
   (10 15) , (10 14) , (21 4) , (16 7) , (15 9)
   Number of bases of rank 2 = 20202250 24

175 aggregates of rank 3 :
   (24 23 11) , (24 23 16) , (24 23 9) , (24 23 3) , (24 22 11)
   (24 22 16) , (24 22 4) , (24 22 2) , (24 19 21) , (24 19 15)
   (24 19 13) , (24 11 21) , (24 11 7) , (24 11 5) , (24 21 16)
   (24 21 9) , (24 21 3) , (24 21 2) , (24 16 15) , (24 16 13)
   (24 16 5) , (24 15 4) , (24 15 3) , (24 15 2) , (24 9 7)
   (24 9 13) , (24 9 5) , (24 7 4) , (24 7 3) , (24 7 2)
   (24 4 13) , (24 4 5) , (24 13 3) , (24 13 2) , (24 5 3)
   (24 5 2) , (23 22 18) , (23 22 10) , (23 22 9) , (23 22 3)
   (23 18 15) , (23 18 14) , (23 10 15) , (23 10 14) , (23 15 3)
   (23 9 14) , (23 14 3) , (22 17 7) , (22 17 6) , (22 8 7)
   (22 8 6) , (22 16 7) , (22 16 6) , (22 7 2) , (22 6 2)
   (19 18 9) , (19 18 6) , (19 17 9) , (19 17 6) , (19 11 15)
   (19 11 13) , (19 15 6) , (19 9 13) , (19 13 6) , (18 17 10)
   (18 17 21) , (18 17 7) , (18 17 5) , (18 10 16) , (18 10 4)
   (18 10 2) , (18 21 16) , (18 21 15) , (18 21 14) , (18 21 2)
   (18 16 9) , (18 16 6) , (18 16 5) , (18 15 7) , (18 15 6)
   (18 15 5) , (18 9 4) , (18 9 14) , (18 9 2) , (18 7 4)
   (18 7 14) , (18 7 2) , (18 4 6) , (18 4 5) , (18 14 6)
   (18 14 5) , (18 6 2) , (18 5 2) , (17 8 4) , (17 8 3)
   (17 21 4) , (17 21 3) , (17 4 5) , (17 5 3) , (11 10 16)
   (11 10 14) , (11 8 16) , (11 8 14) , (11 16 7) , (11 16 5)
   (11 7 14) , (11 14 5) , (10 8 21) , (10 8 13) , (10 21 4)
   (10 21 2) , (10 4 13) , (10 13 2) , (8 21 9) , (8 21 7)
   (8 21 6) , (8 21 3) , (8 16 9) , (8 16 4) , (8 16 6)
   (8 16 3) , (8 15 7) , (8 15 4) , (8 15 6) , (8 15 3)
   (8 9 14) , (8 9 13) , (8 7 14) , (8 7 13) , (8 4 14)
   (8 4 13) , (8 14 6) , (8 14 3) , (8 13 6) , (8 13 3)
   (21 16 9) , (21 16 6) , (21 16 3) , (21 15 3) , (21 9 14)
   (21 9 2) , (21 7 2) , (21 14 3) , (21 6 2) , (21 3 2)
   (16 15 6) , (16 9 13) , (16 9 5) , (16 4 5) , (16 13 6)
   (16 6 5) , (16 5 3) , (15 7 4) , (15 7 3) , (15 7 2)
   (15 4 6) , (15 4 5) , (15 6 3) , (15 6 2) , (15 5 3)
   (9 7 14) , (9 4 13) , (9 14 13) , (9 14 5) , (9 13 2)
   (7 4 14) , (7 4 13) , (7 14 3) , (7 14 2) , (7 13 2)
   (4 14 5) , (4 13 6) , (4 13 5) , (14 5 3) , (13 6 2)
   Number of bases of rank 3 = 459800576

7 aggregates of rank 4 :
   (21 15 7 2) , (21 15 6 2) , (21 14 6 2) , (16 15 4 5) , (14 13 6 3)
   (14 6 5 2) , (13 5 3 2)
   Number of bases of rank 4 = 889192448

0 aggregates of rank 5 :
   Number of bases of rank 5 = 0

0 aggregates of rank 6 :
   Number of bases of rank 6 = 0

Total number of bases is 3385110000
```

A.2.5. Input/Output files for S = B, r = 2 and n = 1

Input

```
3       8
000     4
001     12
010     12
011     36
100     12
101     36
110     36
111     108
```

Output

1 aggregates of rank 1 :
(8)
Number of bases of rank 1 = 108

6 aggregates of rank 2 :
(7 6) , (7 4) , (7 2) , (6 4) , (6 3)
(4 5)
Number of bases of rank 2 = 5184

1 aggregates of rank 3 :
(5 3 2)
Number of bases of rank 3 = 1728

Total number of bases is 7020

A.2.6. Input/Output files for S = B, r = 3 and n = 1

Input

```
10          200
0000000000  4
0000001011  12
0000001101  12
0000001110  12
0000001111  24
0010100001  12
0010101011  36
0010101101  36
0010101111  108
0011000001  12
0011001011  36
0011001101  36
0011001111  108
0011100000  12
0011100001  24
0011101011  108
0011101101  108
0011101110  36
0011101111  288
0100010010  12
0100011011  36
0100011110  36
0100011111  108
0101000010  12
0101001011  36
0101001110  36
0101001111  108
0101010000  12
0101010010  24
0101011011  108
0101011101  36
0101011110  108
0101011111  288
0110110011  36
0110111011  108
0110111111  432
0111000000  12
0111000001  36
0111000010  36
0111000011  144
0111001011  684
0111001101  144
0111001110  144
0111001111  2448
0111010000  36
0111010001  144
0111010010  108
0111010011  432
0111011011  2160
0111011101  540
0111011110  432
0111011111  7668
0111100000  36
0111100001  108
0111100010  144
0111100011  432
0111101011  2160
0111101101  432
0111101110  540
0111101111  7668
0111110000  144
0111110001  432
0111110010  432
0111110011  1260
0111111011  6804
0111111101  1728
0111111110  1728
0111111111  23760
```

```
1000010100  12
1000011101  36
1000011110  36
1000011111  108
1000100100  12
1000101101  36
1000101110  36
1000101111  108
1000110000  12
1000110100  24
1000111011  36
1000111101  108
1000111110  108
1000111111  288
1010100000  12
1010100001  36
1010100100  36
1010100101  144
1010101011  144
1010101101  684
1010101110  144
1010101111  2448
1010110000  36
1010110001  144
1010110100  108
1010110101  432
1010111011  540
1010111101  2160
1010111110  432
1010111111  7668
1011010101  36
1011011101  108
1011011111  432
1011000000  36
1011000001  108
1011000100  144
1011000101  432
1011010011  432
1011011101  2160
1011011110  540
1011011111  7668
1011100000  144
1011110001  432
1011110100  432
1011110101  1260
1011111011  1728
1011111101  6804
1011111110  1728
1011111111  23760
1100010000  12
1100010010  36
1100010100  36
1100010110  144
1100011011  144
1100011101  144
1100011110  684
1100011111  2448
1100100000  36
1100100010  144
1100110100  108
1100110110  432
1100111011  540
1100111101  432
1100111110  2160
1100111111  7668
1101010000  36
1101010010  108
1101010100  144
1101010110  432
1101011011  432
1101011101  540
1101011110  2160
1101011111  7668
1101100110  36
1101101110  108
1101101111  432
1101110000  144
1101110010  432
1101110100  432
1101111010  1260
1101111011  1728
1101111101  1728
1101111110  6804
1101111111  23760
1110110000  144
1110110001  576
1110110010  576
1110110011  2268
1110110100  432
1110110101  1728
```

1110110130	1728
1110130111	6912
1130113011	10692
1130111301	8640
1110111110	8640
1110111111	187488
1111020000	144
1113010001	576
1113010010	432
1111010011	1728
1111010300	576
1111010301	2268
1111010110	1728
1111010111	6912
1111011031	8640
1111011101	10692
1111011130	8640
1113011111	187488
1111100000	144
1111100001	432
1111100010	576
1111100011	1728
1111100100	576
1111100101	1728
1111100110	2268
1111100111	6912
1111101011	8640
1111101301	8640
1111101110	10692
1111101111	187488
1111110000	15228
1111110001	46080
1111110010	46080
1111130011	139428
1111110300	46080
1111110101	139428
1111110110	139428
1111130111	421632
1111111011	740448
1111111101	740448
1111111110	740448
1111111111	12679416

Output

The number of aggregates (classes of B-bases) is *very* large. So, we give only the following table from our results.

Rank	Number of aggregates	Example of aggregates
1	1	(200)
2	4301	(199 198)
3	150973	(195 194 175)
4	230632	(193 191 190 140)
5	40166	(189 181 179 178 89)
6	1496	(177 165 104 92 55 12)
7	9	(134 126 119 91 78 15 4)
Total	*427578*	

REFERENCES

[A1] S.G. Akl, *Parallel sorting algorithms*, Orlando, Academic Press, 1985, pp.29-37.

[A2] T. Aoki, *Dreams for new device-based superchips*, Proc. 23rd IEEE Int. Symp. Multiple-Valued Logic, 1993, pp.140-149.

[A3] T. Aoki, and T. Higuchi, *Impact of interconnection-free biomolecular computing*, Proc.23rd IEEE Int. Symp. Multiple-Valued Logic, 1993, pp.271-276.

[A4] T. Aoki, M. Kameyama, and T. Higuchi, *Interconnection-free biomolecular computing*, IEEE Computer, Vol.25, November 1992, pp.41-50.

[A5] T. Aoki, M. Kameyama, and T. Higuchi, *Design of interconnection-free biomolecular computing system*, Proc. 21st IEEE Int. Symp. Multiple-Valued Logic, 1991, pp.173-180.

[A6] T. Aoki, M. Kameyama, and T. Higuchi, *Interconnection-free set logic network based on a bio-device model*, IEEE Letters, 26, 1990, pp.1015-1016.

[A7] T. Aoki, M. Kameyama, and T. Higuchi, *Design of a highly parallel set logic network based on a bio-device model*, Proc. 19th IEEE Int. Symp. Multiple-Valued Logic, 1989, pp.360-367.

[B1] K.A. Baker and A.F. Pixley, *Polynomial identities and the chinese remainder theorem for algebraic systems*, Math. Z. 143 heft 2, 1975, pp.165-174.

[B2] C. Brink, *Power structures*, RR121, Univ. of Cape Town, and Ph.D. thesis Rand Africaans Univ., 1992.

[C1] M. Conrad, *The lure of molecular computing*, IEEE Spectrum, Vol.23, October 1986, pp.55-60.

[D1] J. Demetrovics, C. Reischer, D.A. Simovici, and I. Stojmenovic, *Enumeration of functions and bases of three-valued set logic under composition with Boolean functions.*, Proc. 24th IEEE Int. Symp. Multiple-Valued Logic, 1994, pp.164-171.

[D2] J. Demetrovics, L. Ronyai, I.G. Rosenberg, and I. Stojmenovic, *On clones and maximal sets in set logic containing all Boolean functions*, Acta Scientarium Mathematicarum, to appear.

[F1] M.A. Franklin, *VLSI performance comparison of banyan and crossbar communications networks*, IEEE Trans. Computers, Vol.C-30, No.4, April 1981, pp.283-291.

[J1] S.V. Jablonskij, *Functional constructions in k-valued logic (Russian)*, Trudy Mat. Inst. Steklov 51, 1958, pp.5-142.

[K1] M. Kameyama, and T. Higuchi, *Prospects of multiple-valued bio-information processing systems*, Proc. 18th IEEE Int. Symp. Multiple-Valued Logic, 1988, pp.237-242.

[K2] T. Kaminuma and G. Matsumoto, *Biocomputers: the next generation from Japan*, Hitachi Information Research Institute, Japan Institute, 1991.

[K3] S.Y. Kung, *VLSI array processors*, IEEE ASSP Magazine, Vol.2, No.3, July 1985, pp.4-22.

[K4] A.V. Kuznecov, *Structures with closure and criteria of functional completeness*, (Russian) Usp. Math. Nauk 16 No2 (98), 1961, pp.201-202.

[L1] R.A. Lerner and A. Tramontano, *Catalitic antibodies*, Scientific American, March 1988, pp.58-70.

[M1] S. Maeda, T. Aoki, and T. Higuchi, *Set-valued logic network based on optical wavelength multiplexing*, Proc. 22nd IEEE Int. Symp. Multiple-Valued Logic, 1992, pp.282-290.

[N1] A. Ngom, C. Reischer and I. Stojmenovic, *Classification of functions and enumeration of bases of set logic under Boolean compositions*, Proc. 25tn IEEE Int. Symp. Multiple-Valued Logic, to appear.

[P1] E. Post, *Introduction to a general theory of elementary propositions*, Amer. J. Math. 43, 1921, pp.163-185.

[P2] N.C. Price and L. Steven, *Fundamentals of enzymology*, Oxford University Press, New York, 1988.

[R1] C. Reischer, and D.A. Simovici, *On the implementation of set-valued non Boolean switching functions*, Proc. 21th IEEE Int. Symp. Multiple-Valued Logic, 1991, pp.166-172.

[R2] C. Reischer, and D.A. Simovici, *Bio-algebras*, Proc. 20th IEEE Int. Symp. Multiple-Valued Logic, 1990, pp.48-53.

[R3] I.G. Rosenberg, *Completeness properties of multiple-valued logic algebra*, in Rine D.C.(ed.), Computer Science and Multiple-Valued Logic: Theory and Application, North-Holland, 1977, pp.144-186.

[R4] I.G. Rosenberg, *The number of maximal closed classes in the set of functions over a finite domain*, Journal of Combinatorial Theory (A) 14, 1973, pp.1-7.

[R5] I.G. Rosenberg, *La structure des fonctions a plusieurs variables sur un ensemble fini*, C.R. Acad. Sci. Paris, A-B, 260, 1965, pp.3817-3819.

[R6] I.G. Rosenberg and D.A. Simovici, *Algebraic aspects of multiple-valued logic*, Proc. 18th IEEE Int. Symp. Multiple-Valued Logic, 1988, pp.266-275.

[R7] S. Rudeanu, *Booleans functions and equations*, North-Holland, Amsterdam, 1974.

[S1] G. Scognamiglio, *Interpolazione per le funzioni algebriche booleane*, Giorn. Mat. Bataglini, 89, 1961, pp.14-41.

[S2] C.L. Seitz, *Concurrent VLSI architectures*, IEEE Trans. Computers, Vol.C-33, No.12, December 1984, pp.1247-1265.

[S3] D.A. Simovici, I. Stojmenovic, and R. Tosic, *Functional completeness and weak completeness in set logic*, Proc. 23rd IEEE Int. Symp. Multiple-Valued Logic, 1993, pp.251-256.

[S4] I. Stojmenovic, *Completeness criteria in many-valued set logic under compositions with Boolean functions*, Proc. 24th IEEE Int. Symp. Multiple-Valued Logic, 1994, pp.177-183.

[S5] I. Stojmenovic, *Some combinatorial and algorithmic problems in many-valued logics*, Inst. Math. Uni. Novi Sad, 1987.

[T1] R. Tosic, I. Stojmenovic, D.A. Simovici, and C. Reischer, *On set-valued functions and Boolean collections*, Proc. 22nd IEEE Int. Symp. Multiple-Valued Logic, 1992, pp.250-254.

[W1] K.C. White, *The prospects for multivalued logic: a technology and applications view*, IEEE Trans. Computers, Vol.C-30, No.9, September 1981, pp.619-634.

[Y1] Y. Yuminaka, T. Aoki, T. Higuchi, *Design of set-valued logic networks for wave-parallel computing*, Proc. 23rd IEEE Int. Symp. Multiple-Valued Logic, 1993, pp.277-282.

[Y2] Y. Yuminaka, T. Aoki, T. Higuchi, *Design of a set logic network based on frequency multiplexing and its applications to image processing*, Proc. 21st IEEE Int. Symp. Multiple-Valued Logic, 1991, pp.8-15.

[Y3] S.Y. Yung, *VLSI array processors*, Englewood Cliffs: Prentice Hall, 1988.