# On the Numbers of Variables to Represent Multi-Valued Incompletely Specified Functions

Tsutomu Sasao
Department of Computer Science and Electronics,
Kyushu Institute of Technology,
Iizuka 820-8502, Japan

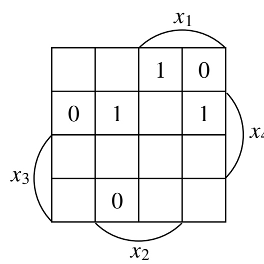


Fig. 1.1. Four-variable incompletely specified logic function.

***Abstract*—In an incompletely specified function $f$, don't care values can be chosen to minimize the number of variables to represent $f$. We consider incompletely specified functions $f : P^n \to Q$, where $P = \{0, 1, \ldots, p-1\}$, $Q = \{0, 1, \ldots, q-1\}$, $u$ combinations are mapped to $i$ $(i = 0, 1, \ldots, q-1)$, $uq = k$, and other combinations are mapped to *don't cares*. We show that most functions can be represented with $2\lceil \log_p(k+1) \rceil$ variables or less. Experimental results are shown to support this.**

## I. Introduction

For completely specified logic functions, logic minimization is a process of reducing the number of products to represent the given function. However, for incompletely specified functions (i.e., functions with *don't cares*), at least two problems exist [5]: The first is to reduce the number of the products to represent the function, and the second is to reduce the number of variables. The first problem is useful for sum-of-products expression (SOP)-based realizations [2], while the second problem is useful for memory-based realizations.

*Example 1.1:* Consider the four-variable function shown in Fig. 1.1, where the blank cells denote *don't cares*. The SOP with the minimum number of products is $\mathcal{F}_1 = x_1 x_4 \vee x_2 \bar{x}_3$, while the SOP with the minimum number of variables is $\mathcal{F}_2 = x_1 x_2 \vee x_1 x_4 \vee x_2 x_4$. Note that $\mathcal{F}_1$ has two products and depends on four variables, while $\mathcal{F}_2$ has three products and depends on only three variables. $x_3$ is a **non-essential variable, since $\mathcal{F}_2$ does not include it.** ■

In this paper, we consider the minimization of the number of variables. Especially, we are interested in the number of variables to represent logic functions whose values are specified for $k$ combinations, where $k$ is small.

Due to the space limitation, all the proofs are omitted.

## II. Definitions and Basic Properties

*Definition 2.1:* A **multi-valued incompletely specified logic function** $f$ is a mapping $D \to Q$, where $D \subset P^n$, $P = \{0, 1, \ldots, p-1\}$, and $Q = \{0, 1, \ldots, q-1\}$.

*Definition 2.2:* $f$ **depends on** $x_i$ if there exists a pair of vectors

$$\begin{aligned} \vec{a} &= (a_1, a_2, \ldots, a_i, \ldots, a_n) \text{ and} \\ \vec{b} &= (a_1, a_2, \ldots, b_i, \ldots, a_n), \end{aligned}$$

such that both $f(\vec{a})$ and $f(\vec{b})$ are specified, and $f(\vec{a}) \neq f(\vec{b})$. If $f$ depends on $x_i$, then $x_i$ is **essential** in $f$, and $x_i$ must appear in every expression for $f$.

*Definition 2.3:* Two functions $f$ and $g$ are **compatible** when the following condition holds: For any $\vec{a} \in P^n$, if both $f(\vec{a})$ and $g(\vec{a})$ are specified, then $f(\vec{a}) = g(\vec{a})$.

*Lemma 2.1:* Let $f_i = f(|x = i)$ for $i = 0, 1, \ldots p-1$. Then, $x$ is **non-essential** in $f$ iff $f_i$ and $f_j$ are compatible for all the pair $(i, j)$.

If $x$ is non-essential in $f$, then $f$ can be represented by an expression without $x$.

*Example 2.1:* Consider the function $f$ in Fig. 1.1. It is easy to verify that $x_1$, $x_2$, and $x_4$ are essential. However, $x_3$ is non-essential. In fact, $f$ is represented as

$$f = x_1 x_2 \vee x_2 x_4 \vee x_1 x_4.$$

■

Essential variables must appear in every expression for $f$, while non-essential variables may appear in some expressions and not in others. Algorithms to represent a given function by using the minimum number of variables have been considered [1], [3], [4], [5].

## III. Analysis for Two-Valued Output Functions

In this section, we derive the number of variables to represent $p$-valued input two-valued output incompletely specified functions. In the analysis that follows, we consider a set of functions (e.g., all incompletely specified functions) restricted by conditions (e.g. the number of *care* values is $k = 2u$).

*Definition 3.1:* A set of functions is **uniformly distributed**, if the probability of occurrence of any function is the same as any other function.

For example, the set of two-valued input two-valued output 4-variable incompletely specified functions with 1 *care* value consists of 32 members, 16 having a single 1 and 16 having a single 0. If the functions are uniformly distributed, the probability of the occurrence of any one of them is $\frac{1}{32}$.

*Theorem 3.1:* Consider a set of uniformly distributed $p$-valued $n$-variable input two-valued output incompletely specified function, where $u$ combinations are mapped to 0, $u$ combinations mapped to 1, and the other $p^n - 2u$ combinations are mapped to *don't cares*. Let $\eta$ be the probability that $f(x_1, x_2, \ldots, x_n)$ can be represented by using only $x_1, x_2, \ldots, x_{t-1}$, and $x_t$, where $t < n$. Then, $\eta > (1 - \tilde{\alpha})^u$, where $\tilde{\alpha} = \frac{u}{p^t}$.

Theorem 3.1 considers the probability for one partition: $X_1 = (x_1, x_2, \ldots, x_t)$ and $X_2 = (x_{t+1}, x_{t+2}, \ldots, x_n)$. However, in practice, we can select a minimum set of variables to represent the function. The following theorem considers such case:

*Theorem 3.2:* Consider a set of uniformly distributed incompletely specified function, where $u$ combinations are mapped to 0, $u$ combinations mapped to 1, and the other $p^n - 2u$ combinations are mapped to *don't cares*. Let $PR$ be the probability that $f(x_1, x_2, \ldots, x_n)$ can be represented by using only $t$ variables. Then, $PR = 1 - \sigma^{\binom{n}{t}}$, where $\sigma = 1 - \eta$, and $\eta$ is the probability that $f(x_1, x_2, \ldots, x_n)$ can be represented by using only $x_1, x_2, \ldots, x_{t-1}$, and $x_t$.

From this, we have the following:

*Conjecture 3.1:* Consider a set of uniformly distributed functions of $n$ variables, where $u$ combinations are mapped to 0, $u$ combinations are mapped to 1, and the other $p^n - 2u$ combinations are mapped to *don't cares*. If

$$t > 2\log_p u - \log_p 4,$$

then more than $95\%$ of the functions can be represented with $t$ variables.

## IV. Analysis for Multiple-Valued Input Index Generation Functions

In practical applications, many functions take multiple values. Here, we consider the class of index generation functions, which are special class of multiple-valued output functions. Such functions have wide applications in pattern matching in the internet [8], [9], [10].

*Definition 4.1:* Consider a set of $k$ different vectors with $n$ components. These vectors are **registered vectors**. For each registered vector, assign a unique integer from 1 to $k$. A **registered vector table** shows the **index** of each registered vector. An **index generation function** produces the corresponding index if the input matches a registered vector, and produces 0 otherwise. $k$ is the **weight** of the index generation function.

In this paper, we assume that $k$ is much smaller than $p^n$, the total number of input combinations.

### A. Number of Variables to Represent Index Generation Functions

In this part, we derive the number of variables to represent an incompletely specified index generation function with $k$ registered vectors. The basic idea is as follows: a function

TABLE 4.1
Registered vector table.

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $f$ |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 2 |
| 0 | 1 | 1 | 1 | 0 | 3 |
| 1 | 0 | 0 | 1 | 1 | 4 |
| 1 | 0 | 0 | 1 | 1 | 5 |
| 1 | 1 | 1 | 1 | 0 | 6 |

TABLE 4.2
Decomposition chart for $f(X_1, X_2)$.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | $x_3$ |
| | | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | $x_2$ |
| | | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | $x_1$ |
| 0 | 0 | | 1 | | | | | | | |
| 0 | 1 | | | 2 | | | 5 | | | |
| 1 | 0 | | | | 3 | | | 6 | | |
| 1 | 1 | | | | | 4 | | | | |
| $x_5$ | $x_4$ | | | | | | | | | |

$f(X_1, X_2)$ is represented by a decomposition chart, where $X_1$ labels the columns and $X_2$ labels the rows. If each column has at most one *care* element, then the function can be represented by using only variables in $X_1$. The next example illustrates this.

*Example 4.1:* Table 4.1 shows a registered vector table consisting of 6 vectors. When no entry matches the input vector, the function produces 0. Consider the decomposition chart shown in Table 4.2. In Table 4.2, $x_1$, $x_2$, and $x_3$ specify the columns, and $x_4$ and $x_5$ specify the rows, and blank elements denote *don't cares*. Note that in Table 4.2, each column has at most one *care* element. Thus, the function can be represented by only the column variables: $x_1$,$x_2$, and $x_3$. ■

From here, we obtain the probability of such a condition by a statistical analysis.

*Theorem 4.1:* Consider a set of uniformly distributed $p$-valued input incompletely specified index generation functions $f(x_1, x_2, \ldots, x_n)$ with weight $k$, where $p \leq k < p^{n-2}$. Let $\eta(k)$ be the probability that $f$ can be represented with $x_1, x_2, \ldots,$ and $x_t$, where $t < n$. Then, $\eta(k) \simeq \exp(-\frac{k^2}{2p^t})$.

The above theorem shows the case when the input variables are removed without considering the property of the function. In practice, we can remove the maximum number of non-essential variables by an optimization program.

*Theorem 4.2:* Consider a set of uniformly distributed incompletely specified index generation functions $f(x_1, x_2, \ldots, x_n)$ with weight $k$, where $p \leq k < p^{n-2}$. Let $PR$ be the probability that $f$ can be represented with $t$ variables, then $PR = 1 - (1 - \eta(k))^{\binom{n}{t}}$, where $\eta(k)$ is the probability that $f$ can be represented with $x_1, x_2, \ldots,$ and $x_t$.

From this, we have the following:

*Conjecture 4.1:* Consider a set of uniformly distributed incompletely specified $p$-valued input $n$-variable index generation functions with weight $k$. If

$$t \geq 2\log_p k - \log_p 4.158,$$

then more than $95\%$ of the functions can be represented with $t$ variables.

Note that there exist functions that require all the variables as shown below. However, the fraction of such functions approaches to zero as $n$ increase.

*Example 4.2:* Consider the $n$-variable incompletely specified index generation function $f$ with weight $k = n + 1$ and $p = 2$:

$$\begin{array}{rcl} f(1,0,0,\ldots,0,0) &=& 1 \\ f(0,1,0,\ldots,0,0) &=& 2 \\ f(0,0,1,\ldots,0,0) &=& 3 \\ \vdots && \vdots \\ f(0,0,0,\ldots,1,0) &=& n-1 \\ f(0,0,0,\ldots,0,1) &=& n \\ f(0,0,0,\ldots,0,0) &=& n+1 \\ f(a_1,a_2,a_3,\ldots,a_{n-1},a_n) &=& d \quad \text{(for other combinations).} \end{array}$$

In this function, all the variables are essential, and no variable can be removed. ■

*Theorem 4.3:* To represent an incompletely specified index generation function with weight $k$, at least $\lceil \log_p(k+1) \rceil$ variables are necessary.

## V. Experimental Results

### A. Random Single-Output Functions

For different $n$ and $p$, we randomly generated 100 functions, where $u$ combinations are mapped to 0, $u$ combinations are mapped to 1, and the remaining $p^n - 2u$ combinations are mapped to *don't cares*. We minimized the number of variables by an exact optimization algorithm [7]. In Table 5.1, the columns headed with *Exp* show average numbers of variables to represent $p$-valued input two-valued output functions, where the set of variables are selected by the optimization algorithm. The values are the average of 100 randomly generated functions. The columns headed by *Conj* show the numbers of variables to represent incompletely specified functions given by Conjecture 3.1. For example, when $p = 2$ and $n = 20$, functions whose 31 minterms are mapped to zeros, 31 minterms are mapped to ones, and the other minterms are mapped to *don't cares*, require, on the average, 6.98 variables to represent the functions. On the other hand, Conjecture 3.1 shows that 8 variables are sufficient. The numbers with * marks show that the bounds were greater than $n$.

TABLE 5.1
AVERAGE NUMBERS OF VARIABLES TO REPRESENT SINGLE-OUTPUT LOGIC FUNCTIONS WITH $\boldsymbol{u}$ 1'S AND $\boldsymbol{u}$ 0'S.

| | $p=2$ $n=20$ | | $p=3$ $n=13$ | | $p=4$ $n=10$ | |
|---|---|---|---|---|---|---|
| $u$ | Exp | Conj | Exp | Conj | Exp | Conj |
| 15 | 4.98 | 6 | 3.88 | 3.79 | 3.11 | 3 |
| 31 | 6.98 | 8 | 4.99 | 5.05 | 4.05 | 4 |
| 63 | 8.98 | 10 | 6.02 | 6.31 | 5.04 | 5 |
| 127 | 10.98 | 12 | 7.50 | 7.57 | 6.06 | 6 |
| 255 | 12.99 | 14 | 8.94 | 8.83 | 7.21 | 8 |
| 511 | 15.09 | 16 | 10.03 | 10.10 | 8.39 | 9 |
| 1023 | 17.50 | 18 | 11.70 | *11.36 | 9.57 | 10 |
| 2047 | 19.70 | 20 | 12.97 | *12.62 | 10.00 | *10 |
| 4095 | 20.00 | *20 | 13.00 | *13* | 10.00 | *10 |

### B. Random Index Generation Functions

We generated uniformly distributed index generation functions. Table 5.2 shows the average numbers of variables to represent $p$-valued input $n$-variables index generation functions with $k$ registered vectors. For the other $p^n - k$ combinations, the outputs are set to *don't cares*. The columns headed with *Exp* show that the average numbers of variables to represent the functions. The columns headed with *Conj* show the number of variables to represent incompletely specified index generation functions with weight $k$ given by Conjecture 4.1. For example, when $k = 31$ and $p = 2$, to represent a uniformly distributed function, experimental results show that, on the average, 6.09 variables are necessary to represent the functions. On the other hand, Conjecture 4.1 shows that 8 variables are sufficient. We performed additional experiments for $p = 3, 5$, and 7 and confirmed that the Conjecture.

TABLE 5.2
AVERAGE NUMBER OF VARIABLES TO REPRESENT INCOMPLETELY SPECIFIED INDEX GENERATION FUNCTION.

| | $p=2$ $n=20$ | | $p=3$ $n=13$ | | $p=4$ $n=10$ | |
|---|---|---|---|---|---|---|
| $k$ | Exp | Conj | Exp | Conj | Exp | Conj |
| 15 | 4.92 | 6 | 3.22 | 3.75 | 3.00 | 3 |
| 31 | 6.09 | 8 | 4.53 | 5.01 | 3.97 | 4 |
| 63 | 8.00 | 10 | 5.79 | 6.23 | 4.94 | 5 |
| 127 | 10.00 | 12 | 7.00 | 7.53 | 5.97 | 6 |
| 255 | 12.00 | 14 | 8.02 | 8.80 | 6.97 | 7 |
| 511 | 14.06 | 16 | 9.50 | 10.06 | 7.94 | 8 |
| 1023 | 16.25 | 18 | 10.97 | 11.32 | 9.05 | 9 |
| 2047 | 18.71 | 20 | 12.37 | 12.58 | 9.98 | 10 |
| 4095 | 19.99 | *20 | 13.00 | *13 | 10.00 | *10 |

## VI. Applications

### A. Two-Valued Case

A terminal access controller (TAC) for a local area network checks whether the requested terminal has permission to access Web, e-mail, FTP, Telnet, or not. Each terminal has its unique MAC address represented by 48 bits. Note that the table for the terminal access controller must be updated frequently.

*Example 6.1:* Let the number of terminals to be connected to a TAC be at most 255. To implement the TAC, we use an IGU and a memory. The number of inputs for the index generation unit (IGU) is 48 and the number of outputs is 8. Fig. 6.1 shows the IGU. When the number of registered vectors is 255, we need about 12 variables to distinguish these vectors. Let $t = 15$ be the number of inputs, and $m = 8$ be the number of outputs of main memory. The size of the main memory is $2^{15} \times 8 = 2^{18} = 256 \times 2^{10}$. The size of the AUX memory is $2^8 \times (48 - 15) = 256 \times 33 = 8448$. Note that, in many cases, only 12 inputs are necessary to distinguish the MAC addresses, but 15 inputs are used to cover more addresses. ■

### B. Four-Valued Case

Deoxyribonucleic acid (DNA) contains the genetic instructions used in the development and functioning of all known

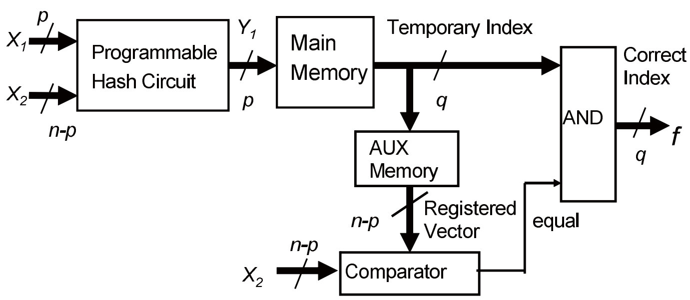


Fig. 6.1. Index generation unit.

TABLE 6.1
INDEX GENERATION FUNCTION

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $f$ |
|---|---|---|---|---|---|---|---|---|
| $A$ | $A$ | $G$ | $A$ | $G$ | $C$ | $T$ | $A$ | 1 |
| $A$ | $A$ | $G$ | $C$ | $A$ | $C$ | $G$ | $C$ | 2 |
| $G$ | $A$ | $A$ | $G$ | $A$ | $T$ | $C$ | $A$ | 3 |
| $C$ | $T$ | $G$ | $G$ | $A$ | $G$ | $G$ | $G$ | 4 |
| $T$ | $A$ | $G$ | $G$ | $G$ | $A$ | $T$ | $A$ | 5 |
| $T$ | $A$ | $T$ | $G$ | $C$ | $C$ | $A$ | $G$ | 6 |
| $T$ | $G$ | $A$ | $C$ | $C$ | $G$ | $C$ | $G$ | 7 |

living organisms and some viruses. The four bases found in DNA are adenine (abbreviated A), cytosine (C), guanine (G) and thymine (T). To represent DNA, we use 4-valued logic.

*Example 6.2:* Consider the circuit to detect DNA patterns shown in Table 6.1. Since each pattern consists of 8 characters, a single-memory realization requires a memory with $2 \times 8 = 16$ inputs. Since, it has three outputs, the memory size is $2^{16} \times 3 = 192 \times 2^{10}$, or 192 kilobits.

However, these patterns can be distinguished by using only two characters: $x_4$ and $x_7$. Fig. 6.2 shows the circuit to detect the DNA patterns.

When the input pattern is $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (G, A, A, G, A, T, C, A)$, the main memory produces 3, since the input to the main memory is $(x_4, x_7) = (G, C)$. Thus, the possible index is 3. Then, the AUX memory produces $(x_1, x_2, x_3, x_5, x_6, x_8) = (G, A, A, A, T, A)$. The comparator verifies that the outputs of the AUX memory is the same as the input pattern $(x_1, x_2, x_3, x_5, x_6, x_8)$. This means that the input vector is registered. Finally, the AND gate produces the index 3.

When the input pattern is $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = (A, A, A, G, A, T, C, A)$. Although this pattern is not in Table 6.1, the main memory produces 3, as a possible index. However, the input pattern $(x_1, x_2, x_3, x_5, x_6, x_8) = (A, A, A, A, T, A)$ is different from the output of the AUX memory. So, the comparator produces 0, and the AND gate produces 0, that indicates the input pattern is not registered. In Fig. 6.2, the total amount of memory is only $4^2 \times 3 + 8 \times 6 \times 2 = 144$ bits. ■

## VII. Conclusions

In this paper, we have derived the number of variables to represent incompletely specified $p$-valued two-valued output functions and index generation functions with weight $k$. Such

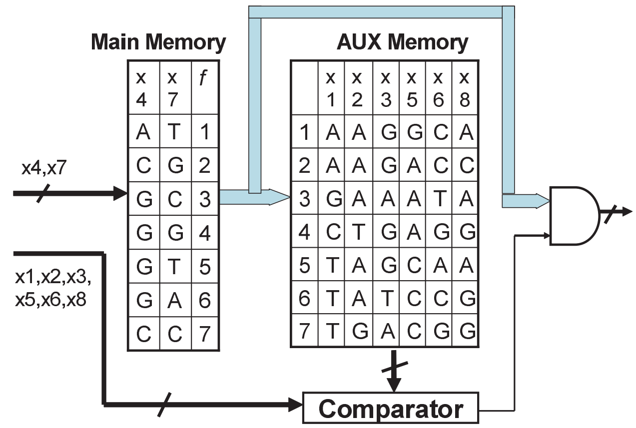


Fig. 6.2. Index generation unit for DNA matching.

functions can be represented by at most $2\lceil \log_p(k+1) \rceil$ variables, in most cases. These results show that reduction of the number of variables is quite effective for incompletely specified functions.


## Acknowledgments

This work was supported in part by the Regional Innovation Cluster Program (Global Type, Second Stage).


## References


[1] F. M. Brown, *Boolean Reasoning: The logic of Boolean Equations*, Kluwer Academic Publishers, Boston, 1990.

[2] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Boston, MA. Kluwer Academic Publishers, 1984.

[3] M. Fujita and Y. Matsunaga, “Multi-level logic minimization based on minimal support and its application to the minimization of look-up table type FPGAs,” *ICCAD-91*, pp. 560-563.

[4] C. Halatsis and N. Gaitanis, “Irredundant normal forms and minimal dependence sets of a Boolean functions,” *IEEE Trans. on Computers*, Vol. C-27, No. 11, pp. 1064-1068, Nov. 1978.

[5] Y. Kambayashi, “Logic design of programmable logic arrays,” *IEEE Trans. on Computers*, Vol. C-28, No. 9, pp. 609-617, Sept. l979.

[6] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.

[7] T. Sasao, “On the number of dependent variables for incompletely specified multiple-valued functions,” *30th International Symposium on Multiple-Valued Logic*, pp. 91-97, Portland, Oregon, U.S.A., May 23-25, 2000.

[8] T. Sasao, “Design methods for multiple-valued input address generators,”(invited paper) *International Symposium on Multiple-Valued Logic* (ISMVL-2006), Singapore, May 2006.

[9] T. Sasao and M. Matsuura, “An implementation of an address generator using hash memories,” *10th EUROMICRO Conference on Digital System Design*, Architectures, Methods and Tools (DSD-2007), Aug. 27 - 31, 2007, Lubeck, Germany, pp.69-76.

[10] T. Sasao, “On the number of variables to represent sparse logic functions,” *ICCAD-2008*, San Jose, California, USA, Nov.10-13, 2008, pp. 45-51.