Fun Exercises for Logic with EXORs and Recursive Programming

Takashi Hirayama

Ver. 2024-02-05

Abstract

This manuscript provides fun exercises for PPRMs and recursive programming. These exercises will help you improve your programming skills.

1 Recursive Algorithm

The following principle[1, 2] formulates the basic classes of AND-EXOR expressions.

Theorem 1 (Expansion Theorem)**.** An arbitrary logic function $f(x_n, x_{n-1}, \ldots, x_1)$ can be expanded as

$$f(x_n, x_{n-1}, \ldots, x_1) = f_0 \oplus x_n f_2 \quad (1)$$
$$f(x_n, x_{n-1}, \ldots, x_1) = \bar{x}_n f_2 \oplus f_1 \quad (2)$$
$$f(x_n, x_{n-1}, \ldots, x_1) = \bar{x}_n f_0 \oplus x_n f_1, \quad (3)$$

where $f_0 = f(0, x_{n-1}, \ldots, x_1)$, $f_1 = f(1, x_{n-1}, \ldots, x_1)$, and $f_2 = f_0 \oplus f_1$. □

Equations (1)–(3) are the **positive Davio expansion**, the **negative Davio expansion**, and the **Shannon expansion**, respectively. By expanding a function f using Equation (1) recursively, we have a logical expression with only un-complemented literals, which is called a PPRM. Another definition of PPRM is given below.

Definition 1. Let x^0 and x^1 denote 1 and x, respectively. The logical expression in the form:

$$\bigoplus_{0 \le i \le 2^n - 1} a_i \cdot x_n^{i_n} x_{n-1}^{i_{n-1}} \cdots x_1^{i_1} \quad (4)$$

is a **positive polarity Reed-Muller expression (PPRM)**, where $(i_n, i_{n-1}, \ldots, i_1)$ is the binary representation of integer i, and $a_i \in \{0, 1\}$ is a constant. □

Equation (4) defines the general form of PPRMs, and Equation (1) explains how to obtain the PPRM of f. A PPRM is also called a Reed-Muller canonical expression or an algebraic normal form. Wikipedia[3] provides a comprehensible summary for it. For a logic function, the PPRM is unique and is a canonical expression. From Definition 1, a PPRM is specified by the bit sequence $a_0, a_1, \ldots, a_{2^n-1}$. In this manuscript, we represent a PPRM as a bit sequence like Example 1.

```
1: function GetPPRM1(b, n): Integer
2:     ▷ Input: b is the truth table of a logic function, and n is the number of variables. ◁
3:     ▷ Output: the RM spectrum of b. ◁
4:     var f0, f1, f2, p0, p2: Integer
5:     if n = 0 then                                   ▷ Terminal condition.
6:         return b
                                                       ▷ Apply Davio expansion recursively.
7:     f1 ← b ≫ 2^(n−1)                                ▷ f1 is the upper half of bit sequence of b
8:     f0 ← b ⊕ (f1 ≪ 2^(n−1))                         ▷ f0 is the lower half of bit sequence of b
9:     f2 ← f0 ⊕ f1
10:    p2 ← GetPPRM1(f2, n − 1)                        ▷ p2 is the upper half of RM spectrum
11:    p0 ← GetPPRM1(f0, n − 1)                        ▷ p0 is the lower half of RM spectrum
12:    return (p2 ≪ 2^(n−1)) ⊕ p0                      ▷ Concatenate p2 and p0.
```

Figure 1: GetPPRM1: an algorithm with the Davio expansion

Example 1. A logic function is often represented by the truth table like Table 1, which corresponds to the minterm expansion: $\bar{x}_3\bar{x}_2\bar{x}_1 \oplus \bar{x}_3\bar{x}_2x_1 \oplus \bar{x}_3x_2x_1 \oplus x_3\bar{x}_2\bar{x}_1 \oplus x_3x_2x_1$. Hereafter, we refer to the output sequence 10011011 as the data structure of truth table of f. Note that, in this manuscript, the output for the input (0,0,0) is the least-significant bit and one for (1,1,1) is the most-significant bit. The same function f can also be represented by the PPRM: $1 \oplus x_2 \oplus x_2x_1 \oplus x_3x_1 \oplus x_3x_2x_1$. Table 2 shows the corresponding a_i's in Definition 1. The bit sequence 10101101 of Table 2 is called the Reed-Muller (RM) spectrum of f, in which $a_{2^{n-1}}$ is the most-significant bit. □

Table 1: Truth table

x_3	x_2	x_1	f	
0	0	0	1	$\bar{x}_3\bar{x}_2\bar{x}_1$
0	0	1	1	$\bar{x}_3\bar{x}_2x_1$
0	1	0	0	$\bar{x}_3x_2\bar{x}_1$
0	1	1	1	$\bar{x}_3x_2x_1$
1	0	0	1	$x_3\bar{x}_2\bar{x}_1$
1	0	1	0	$x_3\bar{x}_2x_1$
1	1	0	0	$x_3x_2\bar{x}_1$
1	1	1	1	$x_3x_2x_1$

Table 2: RM spectrum

i_3	i_2	i_1	a_i	
0	0	0	1	1
0	0	1	0	x_1
0	1	0	1	x_2
0	1	1	1	x_2x_1
1	0	0	0	x_3
1	0	1	1	x_3 x_1
1	1	0	0	x_3x_2
1	1	1	1	$x_3x_2x_1$

From Theorem 1, a conversion algorithm from a truth table to the PPRM can be built with recursive applications of Equation (1). GetPPRM1 in Figure 1 is the pseudo-code of the algorithm. In this pseudo-code, '⊕' is the bit-wise EXOR operation of binary integers. The binary operation $b \gg k$ takes integers b and k, then calculates the integer of shifting b to the right by k bits. '≪' is the left shift version of '≫'.

Exercise 1. Analyze the time complexity of GetPPRM1. For simplicity, assume that '≪', '≫', '⊕', and other primitive operations can be executed in constant time. First give the recurrence relation for the running time $T(n)$ of GetPPRM1(b, n). Then solve the recurrence. Finally, give the order of complexity using big O notation. □

Exercise 2. Implement GetPPRM1 with your favorite programming language. To deal with big integers, a programming language that supports big integers, (e.g., $2^{2^{10}} = 2^{1024}$) should be adopted. Check your program with the test cases given in Table 3. □

Table 3: Test cases for PPRMs

Truth table	n	RM spectrum
0b10011011	3	0b10101101
0b00000001	3	0b11111111
0b10000001	3	0b01111111
0b10110110	3	0b10110110
0b11111111	3	0b00000001
0xD6A3	4	0xA375
0xEC48	4	0xEC48
0x8000	4	0x8000
0xCC006D5F	5	0xFE21FA21
0x5EA3E55C1F1816E4	6	0x2F551D582354BA2C
0xF5EBDD6D7DA16552933A12A2D878C4B3	7	0x5DC6C04E14D8C01FAE65D90F299D0985

2 Iterative Algorithm

Example 2. $\bar{x} = x \oplus 1$ is a basic property of the EXOR operation. By substituting negative literals with this equation, any minterm expansion can be converted to the PPRM. For example, f given in Example 1 is converted to the PPRM by substituting $\bar{x}_3 = x_3 \oplus 1$, $\bar{x}_2 = x_2 \oplus 1$, and $\bar{x}_1 = x_1 \oplus 1$ as follows.

$$\begin{aligned} f &= \bar{x}_3\bar{x}_2\bar{x}_1 \oplus \bar{x}_3\bar{x}_2x_1 \oplus \bar{x}_3x_2x_1 \oplus x_3\bar{x}_2\bar{x}_1 \oplus x_3x_2x_1 \\ &= (x_3 \oplus 1)(x_2 \oplus 1)(x_1 \oplus 1) \oplus (x_3 \oplus 1)(x_2 \oplus 1)x_1 \oplus (x_3 \oplus 1)x_2x_1 \oplus \\ &\quad x_3(x_2 \oplus 1)(x_1 \oplus 1) \oplus x_3x_2x_1 \\ &= 1 \oplus x_2 \oplus x_2x_1 \oplus x_3x_1 \oplus x_3x_2x_1 \end{aligned}$$

□

Based on the idea of Example 2, we can convert a truth table to the RM spectrum. This allows us to make another conversion algorithm without recursive calls. We name the algorithm GetPPRM2 in this manuscript.

Exercise 3. Implement GetPPRM2, and verify your program with the test cases in Table 3. (Hint: you may introduce another data structure suitable for the substitution $\bar{x} = x \oplus 1$, and use it as a temporary and intermediate representation during the conversion from a truth table to the RM spectrum.) In addition, try to minimize the use of **if** statements in your program, by utilizing modern coding style. How many **if** statements did you use to implement GetPPRM2? □

Exercise 4. Write an automated test program to verify whether GetPPRM1 equals GetPPRM2 for randomly generated truth tables. Also, provide the test results. The program should test at least ten random truth tables for each $n = 4, 5, \ldots, 10$. Conforming to the standard unit testing framework for your programming language is preferable. □

Exercise 5. Give the pseudo-code of your GetPPRM2 to express your algorithm clearly, like GetPPRM1 in Figure 1. □

3 Conclusion

Through these exercises, you have noticed that an algorithm that looks simpler for human beings (like Example 2) does not always result in easier programming. Actually, GetPPRM1 with

recursion is far easier for programming than GETPPRM2 even though the conversion process of GETPPRM1 is hardly traced by humans. Recursion is a powerful technique to make programming easier. It is especially effective in the case where the given problems or data structures are defined in recursive forms. Keep it in mind and enjoy programming with recursive techniques!

References

[1] M. Davio, J. P. Deschamps, and A. Thayse. *Discrete and Switching Functions*. McGraw-Hill International, 1978.

[2] T. Sasao. *Switching Theory For Logic Synthesis*. Kluwer Academic Publishers, February 1999.

[3] Wikipedia, Algebraic normal form. `https://en.wikipedia.org/wiki/Algebraic_normal_form`