Comments on “Sympathy: Fast Exact Minimization of Fixed Polarity Reed–Muller Expansion for Symmetric Functions”

Jon T. Butler, Gerhard W. Dueck, Vlad P. Shmerko, and Svetlana Yanuskevich

Abstract—The above paper finds an optimal fixed-polarity Reed–Muller expansion of an one-variable totally symmetric function using an OFDD-based algorithm that requires \(O(n^2)\) time and \(O(n^2)\) storage space. However, an algorithm based on Suprun’s transeunt triangles \([1],[3],[4]\) requires only \(O(n^3)\) time and \(O(n^2)\) storage space. An implementation of this algorithm yields computation times lower by several orders of magnitude.

Index Terms—FPRM (fixed polarity Reed–Muller expressions), two-level AND/EXOR forms, symmetric functions, logic synthesis, minimization.

I. INTRODUCTION

A recent program, Sympathy\(^1\), for finding optimal polarity Reed–Muller (FPRM) expansions of symmetric functions is based on an algorithm whose data structure is an OFDFF of the given function. It requires \(O(n^2)\) operations and \(O(n^3)\) storage space, where \(n\) is the number of variables. However, if one uses a more efficient data structure, specifically the transeunt triangle of Suprun \([1],[3],[4]\), the same computation can be done with \(O(n^3)\) operations and \(O(n^3)\) storage space. The improvement is achieved because coefficients needed in various expansions are computed and stored only once, whereas Sympathy builds a new OFDFF for each polarity. On benchmark functions, the speed improvement is by orders of magnitude.

II. NOTATION

A FPRM expansion for a general function \(f(x_1, x_2, \ldots, x_n)\) is

\[
f(x_1, x_2, \ldots, x_n) = \sum_{i=0}^{2^n-1} c_i x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}
\]

where \(x_i^*\) is either \(x_i\) or \(\overline{x_i}\), everywhere. The term fixed-polarity refers to the fact that each variable occurs in the expression only in one way, \(x_i\) or \(\overline{x_i}\). For example, \(f(x_1, x_2, x_3) = \overline{x_1} x_2 x_3 + x_1 x_2 x_3\) has the following four FPRM expansions.

No variables complemented: \(1 \oplus [x_1 \oplus x_2 \oplus x_3] \oplus [x_1 x_2 \oplus x_1 x_3 \oplus x_2 x_3]\)

One variable complemented: \(\overline{x_1} \oplus [\overline{x_2} x_2 \oplus \overline{x_3} x_3 + x_2 x_3]\)

Two variables complemented: \(\overline{x_1} \oplus [\overline{x_2} x_2 \oplus \overline{x_3} x_3 + x_2 x_3]\)

All variables complemented: \(1 \oplus [\overline{x_1} x_1 \oplus \overline{x_2} x_2 \oplus \overline{x_3} x_3] \oplus [x_1 x_2 \oplus x_1 x_3 \oplus x_2 x_3]\)

Note the total number of product terms required to realize this function. In the first and fourth FPRM expansions, seven terms are required, while in the second and third, only four are required. The **FPRM simplification problem** is to determine which of \(n+1\) polarities (number of complemented variables) yields the FPRM expansion with the fewest terms. In this example, the two middle polarities are both optimum, yielding an expansion of four terms each.

A function \(f(x_1, x_2, \ldots, x_n)\) is (totally) symmetric if and only if it is unchanged by any permutation of variables. For example, \(f(x_1, x_2, x_3) = \overline{x_1} x_2 x_3 + x_1 x_2 x_3\) is symmetric. Certain coefficients in the FPRM expansion of a symmetric function are identical. Let the Reed–Muller expansion matrix of a symmetric function be an \((n+1) \times (n+1)\) matrix of binary coefficients

\[
R_{Mi} = \begin{bmatrix}
0 & 0 & \cdots & 0 & 0 \\
0 & 0 & \cdots & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \cdots & 0 & 0 \\
1 & 1 & \cdots & 1 & 1
\end{bmatrix}
\]

where \(d_{ij}\) is the coefficient of a product term of \(x_i\)’s in an FPRM expansion \((i)\) in which \(j\) variables are complemented and \(k\) are not. For the four FPRM expansions of \(f(x_1, x_2, x_3) = \overline{x_1} x_2 x_3 + x_1 x_2 x_3\), we have

\[
\begin{align*}
R_{M0} &= \begin{bmatrix} 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \end{bmatrix} \\
R_{M1} &= \begin{bmatrix} 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \end{bmatrix} \\
R_{M2} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \end{bmatrix} \\
R_{M3} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \end{bmatrix}
\end{align*}
\]

The bold values represent coefficients in the corresponding FPRM expansion. The \(0\)s not in bold are \(0\) in all Reed–Muller expansion matrices for the same polarity.

A symmetric function is completely specified by a carry vector of logic values \(A = [a_0, a_1, \ldots, a_n]\), such that \(f(x_1, x_2, \ldots, x_n) = a_i\) for all assignments of values to \(x_1, x_2, \ldots, x_n\) that have \(i\) \(1\)s, where \(0 \leq i \leq n\). For example, the carry vector of \(f(x_1, x_2, x_3) = \overline{x_1} x_2 x_3 + x_1 x_2 x_3\) is \([1, 0, 0, 1]\).

III. TRANSEUNT TRIANGLE REPRESENTATION OF REED–MULLER EXPANSIONS

Consider a triangle of \(0\)s and \(1\)s, where the base is a symmetric function’s \(n+1\)-bit carry vector. Immediately below this is a vector of \(n\) \(1\)s and \(0\)s formed by the exclusive OR of adjacent bits in the carry vector. Immediately below this is a vector of \(n-1\) \(1\)s and \(0\)s formed by the exclusive OR of adjacent bits in the previous vector, etc. At the bottom is a single \(1\) or \(0\). Doing this for \(f(x_1, x_2, x_3) = \overline{x_1} x_2 x_3 + x_1 x_2 x_3\) yields the triangle in Fig. 1.

The resulting triangle is the transeunt triangle, originated by Suprun \([3],[4]\). Notice that the bits along the triangle’s left side are coefficients in \(R_{M0}\), while bits along the right side are coefficients in \(R_{M3}\). Additionally, embedded rectangles represent the coefficients in \(R_{M1}\) and \(R_{M2}\). This can be seen in Fig. 2.

---

Comments on 'Sympathy: Fast Exact Minimization of Fixed Polarity Reed-Muller Expansion for Symmetric Functions'

The above paper finds an optimal fixed-polarity Reed?-Muller expansion of an -variable totally symmetric function using an OFDD-based algorithm that requires ( ) time and ( ) storage space. However, an algorithm based on Suprun's transeunt triangles [1], [3], [4] requires only ( ) time and ( ) storage space. An implementation of this algorithm yields computation times lower by several orders of magnitude.
TABLE I
EXECUTION TIMES (IN SECS.) FOR SYMMETRIC BENCHMARK FUNCTIONS*

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>co14</td>
<td>14</td>
<td>0</td>
<td>010(^5)</td>
<td>14</td>
<td>14</td>
<td>368.1</td>
<td>0.3</td>
</tr>
<tr>
<td>m1</td>
<td>6</td>
<td>6</td>
<td>10(^6)</td>
<td>6</td>
<td>1</td>
<td>1.0</td>
<td>0.1</td>
</tr>
<tr>
<td>m4</td>
<td>6</td>
<td>8</td>
<td>10(^6)</td>
<td>6</td>
<td>1</td>
<td>4.2</td>
<td>0.1</td>
</tr>
<tr>
<td>misex2</td>
<td>8</td>
<td>5</td>
<td>0(^2)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>mrej</td>
<td>10</td>
<td>10</td>
<td>1(^{10})</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>0.6</td>
</tr>
<tr>
<td>xor5</td>
<td>2</td>
<td>0</td>
<td>(01)(^3)</td>
<td>0</td>
<td>2.4</td>
<td>5</td>
<td>0.5</td>
</tr>
<tr>
<td>rd53</td>
<td>5</td>
<td>0</td>
<td>0(^4)(^{12})</td>
<td>0</td>
<td>5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>rd53</td>
<td>5</td>
<td>1</td>
<td>0(^2)(^{12}) (^{10})</td>
<td>0</td>
<td>5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>rd53</td>
<td>5</td>
<td>2</td>
<td>(01)(^3)</td>
<td>0</td>
<td>2.4</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>rd73</td>
<td>7</td>
<td>0</td>
<td>0(^4)(^{14})</td>
<td>0</td>
<td>35</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>rd73</td>
<td>7</td>
<td>1</td>
<td>(01)(^2)(^{2})</td>
<td>0</td>
<td>21</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>rd73</td>
<td>7</td>
<td>2</td>
<td>(01)(^4)</td>
<td>0</td>
<td>2.4,4,6</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>rd845</td>
<td>8</td>
<td>0</td>
<td>(0(^2)(^{12})(^{10}))</td>
<td>0</td>
<td>28</td>
<td>4.7</td>
<td>0.1</td>
</tr>
<tr>
<td>rd845</td>
<td>8</td>
<td>1</td>
<td>(01)(^3)(^9)</td>
<td>0</td>
<td>8</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>rd845</td>
<td>8</td>
<td>2</td>
<td>0(^0)(^1)</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>rd845</td>
<td>8</td>
<td>3</td>
<td>0(^4)(^{12})(^{2})</td>
<td>0</td>
<td>70</td>
<td>5.3</td>
<td>0.1</td>
</tr>
<tr>
<td>sym4</td>
<td>4</td>
<td>0</td>
<td>0(^1)(^{2})</td>
<td>4</td>
<td>6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>sym5</td>
<td>6</td>
<td>0</td>
<td>0(^1)(^{2})</td>
<td>0</td>
<td>36</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>sym9</td>
<td>9</td>
<td>0</td>
<td>0(^3)(^{10})</td>
<td>4,5</td>
<td>173</td>
<td>11.7</td>
<td>0.1</td>
</tr>
<tr>
<td>sym10</td>
<td>10</td>
<td>0</td>
<td>0(^4)(^{15})</td>
<td>0</td>
<td>266</td>
<td>27.6</td>
<td>0.2</td>
</tr>
<tr>
<td>sym12</td>
<td>12</td>
<td>0</td>
<td>0(^4)(^{15})</td>
<td>0.12</td>
<td>1288</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>sym15</td>
<td>15</td>
<td>0</td>
<td>0(^4)(^{15})</td>
<td>7.8</td>
<td>15139</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>dbruijn_2</td>
<td>4</td>
<td>0</td>
<td>0(^2)(^{10})</td>
<td>0</td>
<td>6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>dbruijn_3</td>
<td>8</td>
<td>0</td>
<td>0(^2)(^{10})</td>
<td>7</td>
<td>256</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>dbruijn_4</td>
<td>16</td>
<td>0</td>
<td>0(^4)(^{10})(^{2})(^{10})(^{2})</td>
<td>17</td>
<td>106,284</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>dbruijn_5</td>
<td>35</td>
<td>0</td>
<td>0(^{10})(^{1})(^{2})(^{10})(^{2})(^{1})(^{2})(^{1})(^{2})(^{1})(^{2})</td>
<td>6</td>
<td>15,215,790,080</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

IV. THE ALGORITHM AND ITS TIME AND SPACE COMPLEXITY

A. The Algorithm

Note that a single element of the transeunt triangle represents one or more coefficients in the various Reed–Muller expansion matrices. The efficiency of the transeunt triangle is due to the fact that it is not necessary to recompute this coefficient for each polarity.

Algorithm 1 [4]
1) Generate the transeunt triangle.
2) For each \(RM_i\), extract the coefficients \(d_{ij,k}\), and compute the number of product terms.
3) Choose an \(RM_i\) with the fewest product terms.

B. Time and Space Complexity

The following lemma gives both the time and space complexity of the above algorithm. The time complexity is due to [4].

Lemma 4.1: Algorithm 1 is an \(O(n^3)\)-time algorithm that requires \(O(n^2)\) storage space for computing the optimal fixed-polarity Reed–Muller expansion of a symmetric function on \(n\) variables.

Proof: In applying the algorithm, \(O(n^2)\) storage locations are required for the coefficients in the triangles. \(O(n)\) locations are required to store the number of product terms, one for each of the \(n + 1\) polarities, for a total of \(O(n^2)\) locations.

The OFDD approach has time complexity \(O(n^3)\), and space complexity \(O(n^3)\). Thus, Algorithm 1 represents a significant improvement.

V. EXPERIMENTAL RESULTS

Suprun [3], [4] did not apply his algorithm to benchmark functions. Our implementation is called Symphony, (symmetric phunction optimizng system), which is written in C++ and compiled under Microsoft’s Visual Studio Version 6.0 for Windows98. It was run on a 400 MHz. Pentium system.

A. Comparison of Symphony on benchmark functions

Table I shows, for certain symmetric benchmark functions, the execution time of Symphony compared to Sympath and to FDD, another OFDD-based minimizer that does not consider symmetry [2]. Table I also shows the number of inputs (In), the Output Number (Out), the Carrier Vector expressed as a regular expression (Car. Vec.), the polarity(ies) that produced the optimal realization (Opt. Pol.), and the number of product terms in the optimal solution (Products). The three execution times (FDD, Sympathy, and Symphony) are shown in seconds.

As can be seen, Symphony is very fast, requiring no more than 0.0002 secs. on any of the functions considered by Dreschler and Becker. Indeed, these execution times are less than the time interval between real time clock interrupts. As a result, timing functions in C++ return zero elapsed time for program execution. To achieve the necessary resolution, each function was minimized 2,000,000 times and the total time was divided by 2,000,000.

Each dbruijn_k entry in Table I is a d’Bruijn sequence indexed by \(k\). That is, each sequence contains exactly one copy of each of the \(2^k\) binary \(k\)-tuples. Overall, it contains a total of \(2^{2k} + k - 1\) bits. This sequence is such that decision diagram representations for such functions will have many nodes, as there are few repeated subsequences. As a result, algorithms based on decision diagrams will require more computation time than for other symmetric functions.

Table II shows, for certain symmetric functions that are also threshold functions, the relative execution times of FDD, Sympathy, and Symphony. Again, Symphony is fast.

VI. CONCLUSION

Rather than computing the entire FPRM expansion for each polarity, Symphony computes and stores expansion coefficients only once.
using the transeunt triangle, and extracts them, as needed, to form the various expansions. In this way, it achieves a major savings in computation time and storage over Sym\textit{pathy}, which computes a decision diagram for each polarity.

An abbreviated version of \textit{Symphony} can be accessed at http://www.oc.nps.navy.mil/~butler/transeunt.html (word length restrictions on the server preclude carrier vectors with more than 31 bits). Users can input a carrier vector and see the transeunt triangle along with the number of product terms for each polarity.

ACKNOWLEDGMENT

The authors gratefully acknowledge the comments by the editors and three referees which led to improvements.

\begin{table}[h]
\centering
\caption{Execution Times (in secs.) for Symmetric Threshold Functions}
\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
\hline
\texttt{thres9,10} & 10 & 0 & $0^11^2$ & 0 & 11 & 21.0 & 0.1 & 0.000090 \\
\texttt{thres10,11} & 11 & 0 & $0^01^1$ & 0 & 11 & 42.1 & 0.2 & 0.000114 \\
\texttt{thres11,12} & 12 & 0 & $0^11^2$ & 0 & 13 & 87.3 & 0.2 & 0.000143 \\
\texttt{thres12,13} & 13 & 0 & $0^11^2$ & 0 & 13 & 175.7 & 0.2 & 0.000171 \\
\texttt{thres13,14} & 14 & 0 & $0^11^2$ & 0 & 15 & 364.2 & 0.3 & 0.000209 \\
\texttt{thres14,15} & 15 & 0 & $0^11^2$ & 0 & 15 & 756.5 & 0.3 & 0.000251 \\
\texttt{thres15,16} & 16 & 0 & $0^01^2$ & 0 & 17 & 1555.8 & 0.3 & 0.000295 \\
\texttt{thres16,17} & 17 & 0 & $0^11^2$ & 0 & 17 & 3140.3 & 0.4 & 0.000346 \\
\texttt{thres17,18} & 18 & 0 & $0^11^2$ & 0 & 19 & 6483.9 & 0.4 & 0.000406 \\
\texttt{thres18,19} & 19 & 0 & $0^11^2$ & 0 & 19 & 13033.1 & 0.4 & 0.000468 \\
\texttt{thres19,20} & 20 & 0 & $0^11^2$ & 0 & 21 & 25870.1 & 0.4 & 0.000540 \\
\texttt{thres20,21} & 21 & 0 & $0^01^2$ & 0 & 21 & 53549.4 & 0.5 & 0.000612 \\
\texttt{thres21,22} & 22 & 0 & $0^01^2$ & 0 & 23 & \textasciitilde & \textasciitilde & \textasciitilde \\
\texttt{thres22,23} & 23 & 0 & $0^01^2$ & 0 & 23 & \textasciitilde & \textasciitilde & \textasciitilde \\
\texttt{thres23,24} & 24 & 0 & $0^01^2$ & 0 & 25 & \textasciitilde & \textasciitilde & \textasciitilde \\
\texttt{thres24,25} & 25 & 0 & $0^01^2$ & 0 & 25 & \textasciitilde & \textasciitilde & \textasciitilde \\
\texttt{thres25,26} & 26 & 0 & $0^01^2$ & 0 & 27 & \textasciitilde & \textasciitilde & \textasciitilde \\
\texttt{thres26,27} & 27 & 0 & $0^01^2$ & 0 & 27 & \textasciitilde & \textasciitilde & \textasciitilde \\
\texttt{thres27,28} & 28 & 0 & $0^01^2$ & 0 & 29 & \textasciitilde & \textasciitilde & \textasciitilde \\
\texttt{thres28,29} & 29 & 0 & $0^01^2$ & 0 & 29 & \textasciitilde & \textasciitilde & \textasciitilde \\
\texttt{thres29,30} & 30 & 0 & $0^01^2$ & 0 & 31 & \textasciitilde & \textasciitilde & \textasciitilde \\
\hline
\end{tabular}
\end{table}

REFERENCES


