Efficient Algorithms for Minimization of Polynomial Representation of Weakly Specified Boolean Functions and Systems

A. Zakrevskij (Belarus),

L. Zakrevski (USA)

The problem formulation.

• Two-stage AND/EXOR-circuits have known advantages over traditional AND/OR-circuits.

1. better testability.

2. more compact when implementing symmetrical Boolean functions inherent in arithmetic.

• Their optimal synthesis can be reduced to economical representation of Boolean functions by polynomials of some type:

(a) Zhegalkin (mod-2-sum of positive conjunctive terms, $ac \oplus bde \oplus ad$ for example),

(b) Reed-Muller with fixed polarity (where each variable can be presented only in one form, either positive or negated - as in $a^{'}c \oplus bde^{'} \oplus a^{'}d$),

(c) or general Exclusive Sum of Products (ESOP) type.

The considered problem:

Find the shortest polynomial (with the minimum number of terms) which implements a given Boolean function.

1. Many methods suggested for completely specified functions.

2. When the function is not defined on some d input combinations - so called *don't cares* - the problem becomes more difficult.

3. Methods based on generating and checking all 2^d different assignments for don't cares are practical only for small d.

4. The problem is even harder for systems of Boolean functions (multi-output Boolean functions).

• Much interest is recently in weakly specified Boolean functions of many (n) variables defined only on few (m) input combinations, which constitute the definition area D.

• Such functions proved to be very useful in different fields

- logic design,
- knowledge extraction from data,
- pattern recognition,
- etc.

• In this case the area of uncertainty can be very large, containing $d = 2^n - m$ don't cares.

• The number of different assignments for them can exceed 10^{18} for $n = 6$ and small m.

• Therefore, trying all different assignments in order to find the best one is principally unfeasible for $n > 5$.

- Hence another approach is needed for weakly specified Boolean functions, which should ignore don't cares and focus on the area of definition D, where all useful information is contained.

- Such an approach was suggested and developed by these authors.

- It is briefly described below.

- See references in the printed version.

Implementation of a single-output Boolean function.

The problem is reduced to solving a system of m linear logical equations with p variables.

- the equations are put in 1-1 correspondence with elements of the definition area D,

- p is the number of different projections of conjunctive terms onto D, substantially less than the total number of products 2^n in case of Zhegalkin polynomials.

- Consider a Boolean function

$u(x) = u(x_1, \ldots, x_n)$

defined on m inputs (combinations of argument values represented by some Boolean vectors d_i - values of x).

- These inputs constitute the definition area for the function - a set $D = \{d_1, \ldots, d_m\}$.

- This set can be represented by a corresponding $m \times n$ Boolean matrix $\mathbf{D}$ with entries $d_i^j = 1$ iff $x_j = 1$ in d_i: $d_i^j = x_j(d_i)$.

- The values of the function $u(x)$ on this area can be represented by a Boolean vector $u = (u_1, \ldots, u_m)^T : u_i = u(d_i)$.

For example ($x = (a, b, c, d, e)$),

$$\mathbf{D} = \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix} \end{array} \qquad \begin{array}{c} u \\ \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \end{array}$$

- Suppose we are looking for a simplest Zhegalkin polynomial implementing this function.

- All different projections of positive terms onto D are obtained by the conjunctive closure of the set of columns from $\mathbf{D}$

  - this is the repeated operation of multiplace componentwise conjunction of these columns.

- The result is represented by matrix

$\mathbf{R} = \mathbf{cl}(\mathbf{D})$

having the staircase form (columns are marked with related products):

$$
\mathbf{R} = \begin{array}{cccccccccccccc}
1 & a & d & ad & b & c & bc & ab & ac & e & cd & ae & & u \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & & 1 \\
1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & & 1 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & & 1
\end{array}
$$

• A certain linear logical equation corresponds to every row r_i taken together with the component u_i of vector $\mathbf{u}$ shown on the right.

• For example, the first row represents the equation $1 \oplus a \oplus d \oplus ad = 0$.

• Each following row (going from top to bottom) presents an equation involving some new variable - because of the staircase form of the matrix.

• This enables to solve all equations one by one, beginning from the top, and to find in such a way some approximate solution of the system as a whole.

• An exact solution of the problem (finding a shortest implementing polynomial) can be achieved by a modified tree searching technique.

Experiments.

- The proposed algorithms have been tested on PC AT 386, using randomly genetated incomplete Boolean functions.

- Functions generated for various values of n and m.

- The shortest polynomials were looked for.

- For any functions with $n < 12$ and $m < 12$ optimal solutions were found in less than one second.

- Up to $n = 16$ and $m = 20$ three minutes were sufficient.

- In the case of large n and m, an approximate algorithm was applied for finding **suboptimal** solutions.

- It is characterized by the following table

- k is the number of terms,

- t is the time in seconds.

n	50	100	200	300	400	500	600	700
m	1100	900	700	600	500	400	400	400
k	177	114	73	54	42	29	31	31
t	156	166	175	174	153	126	153	168

Implementation of a system of Boolean functions.

• This problem can be reduced to solving a matrix linear logical equation.

• Any shortest polynomial implementation of a system of Boolean functions $u_1, \ldots, u_f$ is presented by a solution of equation

$\mathbf{RX} = \mathbf{U}$, where:

(a) $\mathbf{R} = \mathbf{cl}(\mathbf{B})$,

(b) the columns of $\mathbf{U}$ represent the projections of the functions $u_1, \ldots, u_f$ onto D,

(c) values 1 in the columns of the sought-for matrix $\mathbf{X}$ mark conjunctive terms included in polynomials implementing corresponding output functions.

• The number of non-zero rows of $\mathbf{X}$ must be minimized by that.

A method for finding an exact solution of this problem was suggested.

- Based on the theory of linear spaces.

- It enables to deal with systems consisting of many functions of many variables.

- The algorithm was tested in the range of $n < 17$, $m < 16$ and f up to 16.

- For randomly generated systems composed of 6 Boolean functions of 12 variables, the minimum Zhegalkin polynomial forms were found, having together 8 terms on the average, in about one second.

- It was found that the execution time depends mainly on the difference $m - r(\mathbf{U})$ between the number of rows in $\mathbf{U}$ and the rank of $\mathbf{U}$, which equals f as a rule.

- The smaller is this difference, the smaller is the run time.

Looking for a "superoptimal" solution.

- Let $\mathbf{C}$ be an arbitrary column minor of $\mathbf{R}$.

- $\mathbf{C}$ is a solution if every vector from $\mathbf{U}$ equals the mod-2-sum of some vectors from $\mathbf{C}$ ($\mathbf{U}$ is generated by $\mathbf{C}$).

- It is known that for any system of Boolean functions represented by matrix $\mathbf{U}$ the number $|\mathbf{C}|$ columns in $\mathbf{C}$ cannot be less than $r(\mathbf{U})$, but is greater as a rule.

- That is why $\mathbf{C}$ is called *superoptimal*, or S-solution, when $|\mathbf{C}| = \mathbf{r}(\mathbf{U})$.

• It was proven that if $\mathbf{U}$ generates $\mathbf{C}$ and $r(\mathbf{U}) = \mathbf{r}(\mathbf{C})$ then $\mathbf{C}$ generates $\mathbf{U}$ and hence it is a superoptimal solution.

• Based on this affirmation, the following algorithm was suggested.

Algorithm.

• Let us consecutively obtain and examine elements of **R**, one by one, in order of non-increasing the weights of corresponding conjunctive terms.

• If a current element is generated by $\mathbf{U}$ and is not generated by $\mathbf{P}$, include it in $\mathbf{P}$, empty at the beginning.

• The process terminates successfully when $|\mathbf{P}|$ equals $r(\mathbf{U})$ - in that case $\mathbf{P}$ presents a sought-for superoptimal solution.

• But if $|\mathbf{P}|$ cannot reach $r(\mathbf{U})$, this testifies the absence of such a solution.

The algorithm was tested and turned out to be very efficient.

• It is superfast indeed, that can be seen from the following results of experiments covering the region of parameters up to several hundreds of variables and functions.

• k is the number of terms in a solution,

• w - the sum of their weights,

• t - the time of calculation in seconds.

n=m=f	50	100	150	200	250	300	350	400
k	50	100	149	199	249	299	350	399
w	50	99	257	298	357	442	349	613
t	1.25	11	24	59	107	180	135	423

Generalization to MVL functions.

• The mentioned above methods for finding optimal polynomial implementations of weakly specified Boolean functions were expanded onto multiple-valued logic.

• The latter was interpreted by that as an algebra in Galois field with some prime $p-GF(p)$, where both addition and multiplication are defined as modulo p operations.

• The most efficient turned out to be the generalization of the algorithm for finding superoptimal solutions.

Conclusion.

1. The suggested approach to the problem of optimizing polynomial representations of Boolean and MVL multi-output functions considers specific features of incomplete functions.

2. This enables to reach high efficiency, especially for very weakly specified functions.

3. The continuation of this research seems to have very good perspectives.