BOOM - a Heuristic Boolean Minimizer

Jan Hlavička, Petr Fišer
Department of Computer Science and Engineering, Czech Technical University
Karlovo nám. 13, 121 35 Prague 2
e-mail: hlavicka@fel.cvut.cz, fiserp@fel.cvut.cz

Abstract

We present a two-level Boolean minimization tool (BOOM) based on a new implicant generation paradigm. In contrast to all previous minimization methods, where the implicants are generated bottom-up, the proposed method uses a top-down approach. Thus instead of increasing the dimensionality of implicants by omitting literals from their terms, the dimension of a term is gradually decreased by adding new literals. Unlike most other minimization tools like ESPRESSO, BOOM doesn't use the definition of the function to be minimized as a basis for the solution, thus the original coverage influences the solution only indirectly through the number of literals used.

Most minimization methods use two basic phases introduced by Quine-McCluskey, known as prime implicant (PI) generation and the covering problem solution. Some more modern methods, like ESPRESSO, combine these two phases, reducing the number of PIs to be processed. This approach is also used in BOOM, the search for new literals to be included into a term aims at maximum coverage of the output function.

The function to be minimized is defined by its on-set and off-set, listed in a truth table. Thus the don't care set, often representing the dominant part of the truth table, need not be specified explicitly. The proposed minimization method is efficient above all for functions with a large number of input variables while only few care terms are defined.

The minimization procedure is very fast, hence if the first solution does not meet the requirements, it can be improved in an iterative manner. The method has been tested on several different kinds of problems, like the MCNC standard benchmarks or larger problems generated randomly.

1. Introduction

The problem of two-level minimization of Boolean functions is usually solved for functions with less than 100 variables. This is due to two mutually dependent facts: larger circuits are not so common, and no minimization method for larger problems has been available so far. However, Boolean minimization is by no means limited to the area of switching circuit design, where it was first identified [8]. The new implementation technologies of digital circuits, e.g., multi-level custom design, FPGAs, and above all PLAs, require this minimization in some phase. Minimization problems with a large number of variables are encountered in many modern application areas like design of on-line real-time control systems, design of built-in self-test equipment for VLSI circuits, in the area of artificial intelligence, in software engineering, etc. These problems are mostly characterized by a limited number of input states for which the output value is determined (care states). On the other hand, the number of don't care states then reaches astronomical values, and the quality of a minimization method is thus determined by its ability to take advantage of their existence without enumerating them. An efficient minimization method must in addition be able to cope with the existence of a large number of prime implicants (PIs) of the given function, most of which are not needed for the minimum solution.

Many attempts have been made to increase the size of problems that can be solved by sacrificing absolute minimality and/or modifying the classical two-phase approach (PI generation, covering problem solution) introduced by Quine and McCluskey. Modification of the two-phase approach means, e.g., combining PI generation with the solution of the covering problem (CP) in order to reduce the size of the problem and above all to reduce the number of implicants to be processed. This combination is a characteristic feature of several modern methods, including the well-known ESPRESSO [4, 7] with its later improvements ESPRESSO-EXACT and ESPRESSO-SIGNATURE [9]. This approach is used also in BOOM, namely in the primary implicant generation phase - the "coverage-directed search".

A common feature of all methods proposed so far is obtaining the final solution from an initial solution (which is contained in the function definition) by improvements directed by some objective function. This approach may be of advantage, if the initial solution is favorable, but in some cases it may lead to a blind end, or to an unnecessary waste of time. The method presented here uses an original approach to implicant generation, which is completely independent of the initial solution. The 1-terms found in the function definition are used only indirectly, for guiding the search for literals to be included into the solution, 0-terms are used to determine whether a given term is an implicant. The don't care set, which is usually the largest of all three sets, is thus not consulted at all.

The BOOM (BOOlean Minimizer) approach proposed here combines PI generation with solving the covering problem, leading to a reduction in the total number of PIs generated. However, the principal improvement over the previous methods consists in speeding up PI generation by applying a top-down approach instead of the commonly used bottom-up approach. Several heuristics allowing us to control individual phases of the solution are used in order to meet the quality requirements and runtime limitations. BOOM was programmed in Borland C++ Builder and tested under MS Windows NT.

This paper has the following structure. After a formal problem statement in Section 2, the principles of the proposed method are presented in Section 3. The iterative procedure is then described in Section 4. The results of extensive experimental verification

are evaluated and commented in Section 5, and the time complexity of the proposed algorithm is evaluated in Section 6.

2. The Problem of Boolean Minimization

The problem of two-level minimization will be defined in a usual way [4, 6, 7]. A Boolean function of *n* input variables is defined by a truth table describing the **on-set** F(x_1, x_2, … x_n) and **off-set** R(x_1, x_2, … x_n). Here the on-set (off-set) is the set of terms to which the output value 1 (0) is assigned. Both minterms and terms of higher dimension may be used for defining the on-set and off-set, hence the individual lines of the truth table may contain don't care entries in the input portion. The terms not represented in the input field of the truth table are implicitly assigned don't care values of the output function, i.e., they represent the **don't care set** D(x_1, x_2, … x_n).

We are going to formulate a synthesis algorithm producing a sum-of-products expression $G = g_1+g_2+\ldots+g_t$, where $F \subseteq G \subseteq F+D$ and *t* is minimal. In the case of a set of *m* functions we will minimize the total number of implicants g_i of all functions, while some of them can be used for more output functions. According to this specification, the number of product terms (implicants) is used as a universal quality criterion. This is mostly justified, but it should be kept in mind that the measure of minimality must correspond to the needs of the intended application. ESPRESSO uses the sum of the number of literals and the number of inputs into all output OR gates (also denoted as the output cost). For the BOOM system the minimization criterion may be set as a parameter.

3. Principle of the Method

3.1. BOOM Structure

When minimizing a single-output function, the BOOM system uses the following three phases: 1. Coverage-directed search (generation of implicants). 2. Implicant expansion (generation of prime implicants). 3. Solution of the covering problem.

For multi-output functions, instead of phase 3, phases 4, 5 and 6 are executed: 4. Prime implicant reduction. 5. Solution of the group covering problem. 6. Solution of the covering problem for each output independently.

3.2. Minimization of Single-Output Functions

3.2.1. Coverage-Directed (CD) Search

The idea of confining implicant generation to those really needed gave rise to the CD-search method, which is the most innovative feature of the BOOM system. It consists in a directed search for the most suitable literals that should be added to some previously constructed term in order to convert it into an implicant of the given function. Thus instead of increasing the dimension of an implicant starting from a 1-minterm (or any other 1-term given in the function definition), we reduce the *n*-dimensional hypercube by adding literals to its term, until it becomes an implicant of the given function. This happens at the moment when this hypercube no longer intersects with any 0-term.

The implicant generation method aims at finding a hypercube that covers as many 1-terms as possible. We start by selecting the most frequent input literal from the given on-set. The selected literal describes an *n-1* dimensional hypercube, which may be an implicant, if it does not intersect with any 0-term. If there are some 0-minterms covered, we add one more literal and verify whether the new term already corresponds to an implicant. After each literal selection we temporarily remove from the on-set the terms that cannot be covered by any term containing the selected literal - the terms containing that literal with the opposite polarity. In the remaining on-set we repetitively find the most frequent literal and include it into the previously found product term until it is an implicant. Then we remove from the original on-set the terms covered by this implicant. Thus we obtain a reduced on-set containing only uncovered terms. Now we repeat the procedure from the beginning and apply it to the uncovered terms, selecting the next most frequently used literal, until the next implicant is generated. In this way we generate new implicants, until the whole on-set is covered. The output of this algorithm is a set of product terms covering all 1-terms and intersecting no 0-term.

3.2.2. Implicant Expansion (IE)

The implicants generated during the CD-search need not be prime. To make them prime, we have to increase their size by IE, which means by removing literals (variables) from their terms. When no literal can be removed from the term any more, we get a PI. The expansion of implicants into PIs can be done by several methods differing in complexity and quality of results obtained. We tested several approaches, from the simplest sequential search (which is linear) to the most complex exhaustive (exponential) search.

A **Sequential Search** systematically tries to remove from each term all literals one by one, whereas the first literal is chosen randomly. Every removal is made permanent if no 0-minterm is covered. Only one PI is generated from each implicant, even if it could yield more PIs. A Sequential Search obviously does not reduce the number of product terms. On the other hand, experimental results show that it reduces the number of literals by approximately 25%.

With a **Multiple Sequential Search** we try all possible starting positions within an implicant, which thus expands into several PIs. This method produces more primes than a Sequential Search, while the time complexity is acceptable.

Even the Multiple Sequential Search algorithm cannot expand an implicant into all possible PIs. To do so, an **Exhaustive Implicant Expansion** must be used. Using recursion or queue, all possible literal removals are then tried until all primes are obtained. Unfortunately, the complexity of this algorithm is exponential.

All these expansion strategies have been tested and evaluated from the point of view of runtime and result quality. Finally the Multiple Sequential Search was selected as the best method for standard problems.

Having found a sufficient set of prime implicants, the covering problem is solved. The heuristics used basically correspond to the method suggested, e.g., in [6, 12].

3.3. Minimization of Multi-Output Functions

When minimizing a multi-output function, each of the outputs is first treated separately. After performing the CD-search and IE phases we have a set of PIs sufficient for covering all functions. However, to obtain the minimum solution we may need group implicants, i.e., implicants of more than one output function that are not primes of any. Hence, all obtained primes are tried for reduction by adding some literals. The method of implicant reduction is similar to a CD-search. Literals are repetitively added to each term until there is no chance that the implicant will be used for more functions. We prefer literals that prevent intersecting with most of the 0-terms of all functions. When no further reduction yields any possible improvement, the reduction is stopped, and the implicant is recorded. After assigning implicants to the output functions the group covering problem is solved. Finally, the output reduction, corresponding to the ESPRESSO's MAKE_SPARSE procedure [4, 7], is performed.

4. Iterative Minimization

When selecting the most frequent literal during the CD search, it may happen that two or more literals have the same frequency of occurrence. When no other criterion can be applied to select one literal, the BOOM system chooses at random. Thus there is a chance that repeated application of the same procedure to the same problem would yield different solutions.

The iterative minimization concept takes advantage of the fact that each iteration produces a new set of implicants satisfactory for covering all minterms. The newly created implicants are added to the previous ones and the covering problem is solved using all of them. The set of implicants grows until a maximum reachable set is obtained. The typical growth of the number of PIs as a function of the number of iterations is shown in Fig. 1 (thin line). The values were obtained during the solution of a problem with 20 input variables and 200 minterms. Theoretically, the more primes we have, the better the solution that can be found. In reality, the quality of the final solution improves rapidly during the first few iterations and then remains unchanged, as can be observed in Fig. 1 (thick line).

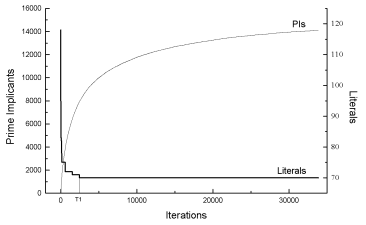

Fig. 1. Growth of PI number and decrease of SOP length during iterative minimization

When the solution meets the requirements, the minimization is stopped.

5. Experimental Results

Many different problems were solved to evaluate the efficiency of the proposed algorithm, especially for large numbers of input variables. The results obtained will be presented in the following subsections. All problems were solved by BOOM and ESPRESSO [14] to compare the results (i.e., number of implicants and/or number of literals and the output cost) and the runtime in seconds. The processor used was a Celeron 433 MHz with 160 MB RAM.

5.1. Standard MCNC Benchmarks

A set of 123 standard MCMC benchmarks [15] was solved by BOOM and ESPRESSO. Of these 123 problems, 51.22 % were solved by BOOM in shorter time than ESPRESSO, in 45.52 % BOOM gave the same result as ESPRESSO (in one case even better). In 30.89 % these results were reached faster than by ESPRESSO. It is also worth mentioning that in 28 cases the BOOM runtime was non-measurable and the timer inserted an implicit value of .01 sec.

The so-called “hard” MCNC benchmarks were also solved by BOOM and ESPRESSO. For 10 problems BOOM found the same solution as ESPRESSO, once in a shorter time, 4 problems gave slightly worse solutions and 5 problems could not be solved because of high memory demands. This is due to the high number of terms, because for BOOM the runtime (and memory demand) grows with the square of the number of terms - see Section 6.

5.2. Problems with more than 100 variables

The MCNC benchmarks have relatively few input variables (only for 3 standard benchmarks does *n* exceed 50). In order to compare the performance of the minimization programs on larger tasks, a set of problems with up to 300 input variables and up to 300 minterms were solved. The truth tables were generated randomly, only the number of input variables, number of care terms and number of don’t cares in the input portion of the truth table (i.e., dimension of a term) were specified. The number of outputs was set equal to 5. The on-set and off-set of each function were kept approximately of the same size. First, the problem was solved by ESPRESSO and then by BOOM, which ran until a solution of the same or better quality was reached. The quality criterion selected was the sum of the number of literals and the output cost. For all samples BOOM found the same or better solution than ESPRESSO in much shorter time. As can be seen from Tab. 1, BOOM needed at most 75 % of the ESPRESSO time, but often its runtime sank below 1 %. The number in parentheses indicates the number of performed iterations.

p/n	100	150	200	250	300
50	1.12 (1)	0.6 (1)	1.38 (4)	0.76 (2)	7.85 (35)
100	1.19 (7)	4.01 (9)	27.08 (35)	8.00 (19)	0.44 (2)
150	11.88 (10)	0.86 (1)	8.51 (20)	14.49 (29)	7.84 (19)
200	18.06 (15)	30.94 (25)	11.34 (20)	0.29 (1)	0.34 (1)
250	74.94 (36)	35.32 (23)	51.56 (50)	12.23 (27)	21.55 (52)
300	60.81 (22)	55.88 (38)	49.11 (34)	25.85 (38)	6.83 (32)

Table 1. Percentage of ESPRESSO runtime needed by BOOM for problems with more than 100 variables

5.3. Solution of Very Large Problems

A third group of experiments aimed at establishing the limits of applicability of BOOM. For this purpose, a set of single-

output functions with up to 1000 input variables and 2000 defined minterms was generated and solved by BOOM. For problems with more than 300 input variables ESPRESSO cannot be used at all. Hence when investigating the limits of applicability of BOOM, it was not possible to verify the results by any other method. The results of this test are listed in Tab. 2, where the time in seconds needed to complete one iteration for various problem sizes is shown. We can see that even the largest problem was solved in less than 5 minutes.

p/n	200	400	600	800	1000
200	0.21	0.38	0.55	0.90	1.06
400	0.98	1.90	3.30	4.84	5.96
600	2.48	4.73	6.94	11.52	18.10
800	4.89	9.76	14.56	24.06	38.58
1000	8.34	15.51	27.88	48.85	74.29
1200	17.64	29.66	42.15	58.37	64.18
1400	23.72	41.49	58.58	74.09	106.65
1600	36.05	73.43	104.90	118.98	161.42
1800	49.53	95.78	146.28	178.29	210.99
2000	60.62	118.39	206.44	204.16	288.87

Table 2. Time for one iteration on very large problems

6. Time Complexity Evaluation

To establish the time complexity of the proposed algorithm, we made a systematic investigation of the dependency of time needed to complete one pass of the algorithm for various sizes of single-output functions. Fig. 2 shows the growth of an average runtime as a function of the number of care minterms (20-300) and of the number of input variables (20-300). The curves on the surface in Fig. 2 indicate that the runtime grows roughly with the square of the number of care minterms and proportionally with the number of input variables.

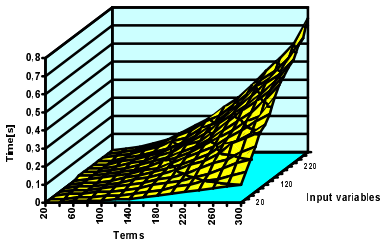

Fig. 2. Dependence of runtime on the number of inputs and number of terms

7. Conclusions

An original Boolean minimization method has been presented. Its most important features are its applicability to functions with several hundreds of input variables and very short minimization times for sparse functions. The function to be minimized is defined by its on-set and off-set, whereas the don't care set, which normally represents the dominant part of the truth table, need not be specified explicitly. The entries in the truth table may be minterms or terms of higher dimensions. The implicants of the function are constructed by reduction of *n*-dimensional cubes; hence the terms contained in the original truth table are not used as a basis for the final solution.

The properties of the BOOM minimization tool were demonstrated on examples. Its application is advantageous above all for problems with large dimensions and a large number of don't care states where it beats other methods, like ESPRESSO, both in minimality of the result and in runtime. The PI generation method is very fast, hence it can easily be used in an iterative manner. The problems with more than 100 input variables were in all cases solved faster and mostly with better results than by ESPRESSO. The dimension of the problems solved by BOOM can easily be increased over 1000, because the runtime grows linearly with the number of input variables. For problems of very high dimension, success largely depends on the size of the care set. This is due to the fact that the runtime grows roughly with the square of the size of the care set.

The BOOM system is available on [13].

References

[1] Fišer, P. - Hlavička, J.: Efficient Minimization Method for Incompletely Defined Boolean Functions, Proc. 4th Int. Workshop on Boolean Problems, Freiberg (Germany), Sept. 21-22, 2000, pp. 91-98

[2] Fišer, P. - Hlavička, J.: Implicant Expansion Method used in the BOOM Minimizer. Proc. IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop (DDECS'01), Gyor (Hungary), 18-20.4.2001, pp. 291-298

[3] Hlavička, J. - Fišer, P.: A Heuristic method of two-level logic synthesis. Proc. The 5th World Multiconference on Systemics, Cybernetics and Informatics SCI'2001, Orlando, Florida (USA) 22-25.7.2001, pp. 283-288, II

[4] Brayton, R.K. et al.: Logic minimization algorithms for VLSI synthesis. Boston, MA, Kluwer Academic Publishers, 1984, 192 pp.

[5] Coudert, O. - Madre, J.C.: Implicit and incremental computation of primes and essential primes of Boolean functions, In Proc. of the Design Automation Conf. (Anaheim, CA, June 1992), pp. 36-39

[6] Coudert, O.: Two-level logic minimization: an overview. Integration, the VLSI journal, 17-2, pp. 97-140, Oct. 94

[7] Hachtel, G.D. - Somenzi, F.: Logic synthesis and verification algorithms. Boston, MA, Kluwer Academic Publishers, 1996, 564 pp.

[8] McCluskey, E.J.: Minimization of Boolean functions. The Bell System Technical Journal, 35, No. 5, Nov. 1956, pp. 1417-1444

[9] McGeer, P. et al.: ESPRESSO-SIGNATURE: A new exact minimizer for logic functions. Proc. DAC'93

[10] Nguyen, L. – Perkowski, M. – Goldstein, N.: Palmini – fast Boolean minimizer for personal computers. In Proc. DAC'87, pp.615-621

[11] Rudell, R.L. – Sangiovanni-Vincentelli, A.L.: Multiple-valued minimization for PLA optimization. IEEE Trans. on CAD, 6(5): 725-750, Sept.1987

[12] Rudell, R.L.: Logic Synthesis for VLSI Design, PhD Thesis, UCB/ERL M89/49, 1989

[13] http://cs.felk.cvut.cz/~fiserp/boom/

[14] http://eda.seodu.co.kr/~chang/ download/espresso/

[15] ftp://ic.eecs.berkeley.edu