Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
العربية
Deutsch
Español
فارسی
Français
한국어
Italiano
日本語
Polski
Português
Русский
Српски / srpski
ไทย
Türkçe
Українська
中文
Edit links

Quine–McCluskey algorithm

From Wikipedia, the free encyclopedia

The **Quine–McCluskey algorithm** (or **the method of prime implicants**) is a method used for minimization of boolean functions that was developed by W.V. Quine and extended by Edward J. McCluskey.[1][2][3] It is functionally identical to Karnaugh mapping, but the tabular form makes it more efficient for use in computer algorithms, and it also gives a deterministic way to check that the minimal form of a Boolean function has been reached. It is sometimes referred to as the tabulation method. A Quine-McCluskey Solver web implementation was created by Hatem Hassan at The American University of Cairo.

The method involves two steps:

1. Finding all prime implicants of the function.
2. Use those prime implicants in a *prime implicant chart* to find the essential prime implicants of the function, as well as other prime implicants that are necessary to cover the function.

Contents [hide]

Complexity [edit]

Although more practical than Karnaugh mapping when dealing with more than four variables, the Quine–McCluskey algorithm also has a limited range of use since the problem it solves is NP-hard: the runtime of the Quine–McCluskey algorithm grows exponentially with the number of variables. It can be shown that for a function of *n* variables the upper bound on the number of prime implicants is $3^n/n$. If $n = 32$ there may be over 6.5 * 10^{15} prime implicants. Functions with a large number of variables have to be minimized with potentially non-optimal heuristic methods, of which the Espresso heuristic logic minimizer is the de facto standard.[4]

Example [edit]

Step 1: finding prime implicants [edit]

Minimizing an arbitrary function:

$$f(A,B,C,D)=\sum m(4,8,10,11,12,15)+d(9,14).$$

This expression says that the output function f will be 1 for the minterms 4,8,10,11,12 and 15 (denoted by the 'm' term). But it also says that we don't care about the output for 9 and 14 combinations (denoted by the 'd' term). ('x' stands for don't care).

	A	B	C	D		f
m0	0	0	0	0		0
m1	0	0	0	1		0
m2	0	0	1	0		0
m3	0	0	1	1		0
m4	0	1	0	0		1
m5	0	1	0	1		0
m6	0	1	1	0		0
m7	0	1	1	1		0
m8	1	0	0	0		1
m9	1	0	0	1		x
m10	1	0	1	0		1
m11	1	0	1	1		1
m12	1	1	0	0		1

m13	1	1	0	1		0
m14	1	1	1	0		x
m15	1	1	1	1		1

One can easily form the canonical sum of products expression from this table, simply by summing the minterms (leaving out don't-care terms) where the function evaluates to one:

$$f_{A,B,C,D} = A'BC'D' + AB'C'D' + AB'CD' + AB'CD + ABC'D' + ABCD,$$

which is not minimal. So to optimize, all minterms that evaluate to one are first placed in a minterm table. Don't-care terms are also added into this table, so they can be combined with minterms:

Number of 1s	Minterm	Binary Representation
1	m4	0100
	m8	1000
2	m9	1001
	m10	1010
	m12	1100
3	m11	1011
	m14	1110
4	m15	1111

At this point, one can start combining minterms with other minterms. If two terms vary by only a single digit changing, that digit can be replaced with a dash indicating that the digit doesn't matter. Terms that can't be combined any more are marked with a "*". When going from Size 2 to Size 4, treat '-' as a third bit value. For instance, -110 and -100 or -11- can be combined, but -110 and 011- cannot. (Trick: Match up the '-' first.)

Number of 1s	Minterm	0-Cube	Size 2 Implicants	Size 4 Implicants
1	m4	0100	m(4,12) -100*	m(8,9,10,11) 10--*
	m8	1000	m(8,9) 100-	m(8,10,12,14) 1--0*
	--	--	m(8,10) 10-0	--
	--	--	m(8,12) 1-00	--
2	m9	1001	m(9,11) 10-1	m(10,11,14,15) 1-1-*
	m10	1010	m(10,11) 101-	--
	--	--	m(10,14) 1-10	--
	m12	1100	m(12,14) 11-0	--
3	m11	1011	m(11,15) 1-11	--
	m14	1110	m(14,15) 111-	--
4	m15	1111	--	--

Note: In this example, none of the terms in the size 4 implicants table can be combined any further. Be aware that this processing should be continued otherwise (size 8 etc.).

Step 2: prime implicant chart [edit]

None of the terms can be combined any further than this, so at this point we construct an essential prime implicant table. Along the side goes the prime implicants that have just been generated, and along the top go the minterms specified earlier. The don't care terms are not placed on top - they are omitted from this section because they are not necessary inputs.

	4	8	10	11	12	15	=>	A	B	C	D
m(4,12)*	X				X		=>	-	1	0	0
m(8,9,10,11)		X	X	X			=>	1	0	-	-
m(8,10,12,14)		X	X		X						
m(10,11,14,15)*			X	X		X	=>	1	-	1	-

To find the essential prime implicants, we run along the top row. We have to look for columns with only 1 star. If a column has only 1 star, this means that the minterm can only be covered by 1 prime implicant. This prime implicant is *essential*. For example: in the first column, with minterm 4, there is only 1 star. This means that m(4,12) is essential. So we place a star next to it. Minterm 15 also only has 1 star. This means that m(10,11,14,15) is also essential. Now all columns with 1 star are covered.

The second prime implicant can be 'covered' by the third and fourth, and the third prime implicant can be 'covered' by the second and first, and neither is thus essential. If a prime implicant is essential then, as would be expected, it is necessary to include it in the minimized boolean equation. In some cases, the essential prime implicants do not cover all minterms, in which case additional procedures for chart reduction can be employed. The simplest "additional procedure" is trial and error, but a more systematic way is Petrick's Method. In the current example, the essential prime implicants do not handle all of the minterms, so, in this case, one can combine the essential implicants with one of the two non-essential ones to yield one equation:

$$f_{A,B,C,D} = BC'D' + AB' + AC$$

Both of those final equations are functionally equivalent to the original, verbose equation:

$$f_{A,B,C,D} = A'BC'D' + AB'C'D' + AB'C'D + AB'CD' + AB'CD + ABC'D' + ABCD' + ABCD.$$

See also [edit]

- Boolean algebra (logic)
- Circuit minimization
- Karnaugh map
- Espresso heuristic minimization program
- Petrick's method
- Willard Van Orman Quine
- Buchberger's algorithm (analogous algorithm for algebraic geometry)

References [edit]

1. ^ Quine, W. V. (Oct 1952). "The Problem of Simplifying Truth Functions". *The American Mathematical Monthly* **59** (8): 521–531. doi:10.2307/2308219. Retrieved 25 August 2014.
2. ^ Quine, W. V. (Nov 1955). "A Way to Simplify Truth Functions". *The American Mathematical Monthly* **62** (9): 627–631. doi:10.2307/2307285. Retrieved 25 August 2014.
3. ^ McCluskey, E.J., Jr. (November 1956). "Minimization of Boolean Functions". *Bell System Technical Journal* **35** (6): 1417–1444. doi:10.1002/j.1538-7305.1956.tb03835.x. Retrieved 24 August 2014.
4. ^ Nelson, Victor P. et al. (1995). *Digital Logic Circuit Analysis and Design*. Prentice Hall. p. 234. Retrieved 26 August 2014.

External links [edit]

- Quine-McCluskey Solver, by Hatem Hassan.
- Quine-McCluskey algorithm implementation with a search of all solutions, by Frédéric Carpon.
- All about Quine-McClusky, article by Jack Crenshaw comparing Quine-McClusky to Karnaugh maps
- Karma 3, A set of logic synthesis tools including Karnaugh maps, Quine-McCluskey minimization, BDDs, probabilities, teaching module and more. Logic Circuits Synthesis Labs (LogiCS) - UFRGS, Brazil.
- A. Costa BFunc, QMC based boolean logic simplifiers supporting up to 64 inputs / 64 outputs (independently) or 32 outputs (simultaneously)
- Python Implementation by Robert Dick, with an optimized version.
- Python Implementation for symbolically reducing Boolean expressions.
- Quinessence, an open source implementation written in Free Pascal by Marco Caminati.
- QCA an open source, R based implementation used in the social sciences, by Adrian Duşa
- A series of two articles describing the algorithm(s) implemented in R: first article[*dead link*] and second article[*dead link*]. The R implementation is exhaustive and it offers complete and exact solutions. It processes up to 20 input variables.
- minBool an implementation by Andrey Popov.
- QMC applet, an applet for a step by step analyze of the QMC- algorithm by Christian Roth
- C++ implementation SourceForge.net C++ program implementing the algorithm.
- Perl Module by Darren M. Kulp.
- Tutorial Tutorial on Quine-McCluskey and Petrick's method (pdf).
- Petrick C++ implementation (including Petrick) based on the tutorial above
- C program Public Domain console based C program on SourceForge.net.
- Tomaszewski, S. P., Celik, I. U., Antoniou, G. E., "WWW-based Boolean function minimization" INTERNATIONAL JOURNAL OF APPLIED MATHEMATICS AND COMPUTER SCIENCE, VOL 13; PART 4, pages 577-584, 2003.
- For a fully worked out example visit: http://www.cs.ualberta.ca/~amaral/courses/329/webslides/Topic5-QuineMcCluskey/sld024.htm
- An excellent resource detailing each step: Olivier Coudert "Two-level logic minimization: an overview" INTEGRATION, the VLSI journal, 17-2, pp. 97–140, October 1994
- The Boolean Bot: A JavaScript implementation for the web: http://booleanbot.com/
- open source gui QMC minimizer

Categories: Boolean algebra | Willard Van Orman Quine