Perancangan Perangkat Lunak Pembelajaran Untuk Penyederhanaan Fungsi Boolean Dengan Metode Quine-McCluskey

Liskedame Yanti Sipayung, ST, M.Kom

Dosen Prodi informatika, Fakultas Teknik Industri
Institut Sains dan Teknologi TD Pardede Medan

Email

ABSTRAK

Tiga cara untuk menyederhanakan fungsi boolean adalah aljabar berdasarkan metode hukum aljabar boolean, metode Peta *Karnaugh* dan metode tabulasi *Quine-Mc Cluskey*. Fungsi boolean dengan lebih dari enam variabel akan lebih rumit untuk dilakukan dan perlu membuat software untuk mempermudah fungsi boolean. Untuk membuat perangkat lunak menggunakan metode Quine-Mc Cluskey karena karakter mekanistik dan cocok untuk mempermudah metode boolean dengan jumlah variabel.

Pemrograman Languange yang digunakan untuk sementara adalah *Microsoft Visual Basic*. Perangkat lunak tambahan yang digunakan untuk menyederhanakan dengan *Quine-Mc Cluskey* bermaksud sebagai perangkat bantu bagi mereka yang belajar menyederhanakan fuction boolean dengan *Quine-Mc Cluskey*. Perangkat lunak dapat diselesaikan dengan menyederhanakan langkah fungsi boolean dengan jumlah variabel yang ditentukan.

Kata Kunci: Aljabar Boolean, Fungsi Boolean, Penyederhanaan, Metode Quine-Mc Cluskey, Microsoft Visual Basic.

1.1 Latar Belakang

Fungsi boolean banyak mengandung operasi-operasi yang tidak perlu, literal atau suku-suku yang berlebihansehingga perludilakukannya penyederhanaan atau minimalisasi fungsi boolean. Menyederhanakan fungsi boolean artinya mencari bentuk fungsi lain yang ekivalen tetapi dengan jumlah literal atau operasi yang lebih sedikit. Untuk menyederhanakan fungsi boolean, dapat dilakukan dengan tiga cara yaitu dengan cara aljabar yaitu berdasarkan hukum atau teorema aljabar boolean, metode peta karnaugh, dan metode tabulasi dari Quine-Mc Cluskey.

Dalam penelitian ini, penulis bermaksud untuk membahas metode minimalisasi fungsi boolean menggunakan Quine-McCluskey serta membangun sebuah perangkat lunak bisa menunjukkan langkah-langkah penyederhanaan fungsi boolean dengan metode Quine-Mc Cluskey. Hal ini bertujuan untuk membantu mereka yang sedang mempelajari penyederhanaan fungsi boolean dengan metode Quine-Mc Cluskey.Algoritma Quine-Mc Cluskey adalah metode yang digunakan untuk minimalisasi fungsi boolean. Metode ini pertama kali dikembangkan oleh W.V. Quine dan Edward J.Mc Cluskey.Metode ini sangat cocok jika jumlah peubah dalam fungsi boolean lebih dari enambuah, sedangkan jika menggunakan metode peta karnaugh dengan jumlah peubah lebih dari enam akan menjadi semakin rumit, sebab ukuran peta akan semakin besar. Selain itu, metode peta karnaugh lebih sulitdiprogram dengan komputer karena diperlukan pengamatan visual secara langsung untuk mengidentifikasi minterm–minterm yang akan dikelompokkan. Karena sebab itulah metode Quine-Mc Cluskey di pilih dalam pembuatan perangkat lunak komputer untuk minimalisasi karena sifatnya yang mekanistik dan cocok untuk penyederhanaan fungsi boolean dengan jumlah sembarang peubah. Metode Quine-McCluskey biasa juga disebut dengan metode tabulasi. Metode ini terdiri dari dua langkah penyelesaian, yaitu:term-term sebagai kandidat (Prime Implicant). Dan memilih prime implicant untuk menentukan ekspresi dengan jumlah literal sedikit **(Wahyu Nugraha (2017, Desember). Jurnal Khatulistiwa Informatika:"PerancanganAplikasiPenyederhanaanFungsi Boolean Dengan MetodeQuine-MC Cluskey", *Vol*(2), 145-151).**

Berdasarkan uraian di atas, penulis bermaksud merancang suatu perangkat lunak dan menuangkan dalam skripsi yang berjudul **"Perancangan Perangkat Lunak Pembelajaran Untuk Penyederhanaan Fungsi Boolean Dengan Metode Quine-McCluskey".**

1.2 Rumusan Masalah

Dari penjelasan latar belakang maka penulis merumuskan masalah pada tulisan ini sebagai berikut:

1. Input berupa nomor-nomor *term* dalam bentuk SOP (*Sum Of Product*) atau bentuk POS (*Product Of Sum),* sistem akan memeriksa validitas data *input* (jumlah peubah, simbol peubah dan batas nomor term).
2. Apabila data *input* valid, maka sistem akan melakukan langkah-langkah minimisasi (7 langkah) terhadap data tersebut sesuai dengan metode Quine-McCluskey.
3. Output sistem berupa hasil minimisasi fungsi Boolean dan hasil eksekusi setiap langkah terhadap data *input* hingga didapatkan *output* dalam bentuk SOP dan POS.

1.3 Batasan Masalah

Berdasarkan rumusan masalah di atas maka penulis membatasi masalah yang di atas adalah sebagai berikut :

1. *Input* berupa nomor-nomor *term* dalam bentuk SOP (S*umOf Product)* atau POS *(Product Of Sum).*
2. Jumlah peubah dibatasi maksimum 10 buah atau jumlah suku pada ekspresi Boolean maksimum = 2^{10} buah.
3. Variabel peubah pada fungsi Boolean dapat di-*input.*
4. Perangkat lunak menyajikan tahapan-tahapan minimisasi terhadap *input* fungsi Boolean.
5. *Output* akhir perangkat lunak berupa hasil minimisasi fungsi Boolean dalam bentuk SOP *(Sum Of Product)* atau POS (*Product Of Sum).*

1.4 Tujuan Dan Manfaat

Tujuan penyusunan tugas akhir ini adalah untuk menghasilkan suatu perangkat lunak yang dapat membantu pemahaman tentang minimisasi fungsi Boolean dengan menggunakan metode Quine-McCluskey. Manfaat dari penyusunan tugas akhir ini adalah :

1. Membantu pemahaman terhadap prosedur minimisasi fungsi Boolean dengan metode Quine-McCluskey.
2. Untuk memahami penyederhanaan fungsi boolean dengan metode Quine-McClusky.

II. Landasan Teori

a. Definisi Aljabar *Boolean*

(Definisi 2.1–Menurut Lipschutz, Seymour & Marc Lars Lipson dalam bukunya '*2000 Solved Problems in Discrete Mathematics*', McGraw-Hill, 1992) Misalkan *B* adalah himpunan yang didefinisikan pada dua operator biner, + dan, dan sebuah operator uner,'. Misalkan 0 dan 1 adalah dua elemen yang berbeda dari *B*. Maka, tupel <*B*, +, ., ', 0, 1> disebut aljabar *boolean* jika untuk setiap *a*, *b*, *c*0*B* berlaku aksioma (sering dinamakan juga Postulat Huntington) berikut :

1. Identitas
   (i) $a + 0 = a$
   (ii) $a \,.\, 1 = a$
2. Komutatif
   (i) $a + b = b + a$
   (ii) $a \,.\, b = b \,.\, a$
3. Distributif
   (i) $a \,.\, (b + c) = (a \,.\, b) + (a \,.\, c)$
   (ii) $a + (b \,.\, c) = (a + b) \,.\, (a + c)$
4. Komplemen
   Untuk setiap *a*0*B* terdapat elemen unik *a*' 0*B* sehingga
   (i) $a + a' = 1$
   (ii) $a \,.\, a' = 0$

Elemen 0 dan 1 adalah dua elemen unik yang berada di dalam *B.* 0 disebut elemen terkecil dan 1 disebut elemen terbesar. Kedua elemen unik dapat berbeda – beda pada beberapa aljabar *Boolean* (misalnya ι dan *U* pada himpunan, *False* dan *True* pada proposisi), namun secara umum kita tetap menggunakan 0 dan 1 sebagai dua elemen unik yang berbeda. Elemen 0 disebut elemen *zero*, sedangkan elemen 1 disebut elemen *unit*. Operator + disebut operator penjumlahan, . disebut operator perkalian, dan ' disebut operator komplemen.

b. Aljabar *Boolean* Dua-Nilai

Aljabar *boolean* yang terkenal dan memiliki terapan yang luas adalah aljabar *boolean* dua-nilai (*two-valued boolean algebra*). Aljabar *boolean* dua-nilai didefinisikan pada sebuah himpunan *B* dengan dua buah elemen 0 dan 1 (sering dinamakan *bit* – singkatan dari *binary digit*), yaitu B = {0, 1}, operator biner, + dan . operator uner, '. Kaidah untuk operator biner dan operator uner ditunjukkan pada Tabel 2.1, 2.2, dan 2.3 di bawah ini.

Tabel 2.1 Tabel kaidah operasi

a	*b*	*a . b*
0	0	0
0	1	0
1	0	0
1	1	1

Tabel 2.2 Tabel kaidah operasi +

a	*b*	*a + b*
0	0	0
0	1	1
1	0	1
1	1	1

Tabel 2.3 Tabel kaidah operasi '

A	*a*'
0	1
1	0

Liskedame Yanti Sipayung, ST, M.Kom
Perancangan Perangkat Lunak Pembelajaran Untuk Penyederhanaan Fungsi Boolean Dengan Metode Quine-McCluskey
Halaman 50-57

Kita harus memperhatikan bahwa keempat aksioma di dalam definisi 2.1 terpenuhi pada himpunan $B = \{0, 1\}$ dengan dua operator biner dan satu operator uner yang didefinisikan di atas.

1. Identitas : jelas berlaku karena tabel dapat kita lihat bahwa :
   (i) $0 + 1 = 1 + 0 = 1$
   (ii) $1 . 0 = 0 . 1 = 0$
   yang memenuhi elemen identitas 0 dan 1 seperti yang didefinisikan pada postulat Huntington.
2. Komutatif : jelas berlaku dengan melihat simetri tabel operator biner.
3. Distributif :
   (i) $a . (b + c) = (a . b) + (a . c)$ dapat ditunjukkan benar dari tabel operator biner di atas, dengan membentuk tabel kebenaran untuk semua nilai yang mungkin dari *a*, *b*, dan *c* (Tabel 7.4). Oleh karena nilai – nilai pada kolom $a . (b + c)$ sama dengan nilai – nilai pada kolom $(a . b) + (a . c)$, maka kesamaan $a . (b + c) = (a . b) + (a . c)$ adalah benar.
   (ii) Hukum distributif $a + (b . c) = (a + b) . (a + c)$ dapat ditunjukkan benar dengan membuat tabel kebenaran dengan cara yang sama seperti (i).

Tabel 2.4 Tabel kebenaran a . (b + c) = (a . b) + (a . c)

a	b	C	b + c	a . (b + c)	a . b	a . c	(a . b) + (a . c)
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

http://nasrulyamand.blogspot.com/2012/04/materi-kuliahhai-kawan.html

4. Komplemen : jelas berlaku karena Tabel 2.4 memperlihatkan bahwa :
   (i) $a + a' = 1$, karena $0 + 0' = 0 + 1 = 1$ dan $1 + 1' = 1 + 0 = 1$
   (ii) $a . a = 0$, karena $0 . 0' = 0 . 1$ dan $1 . 1' = 1 . 0 = 0$

Karena keempat aksioma terpenuhi, maka terbukti bahwa $B = \{0 , 1\}$ bersama – sama dengan operator biner + dan ., operator komplemen ‘ merupakan aljabar *Boolean*. Untuk selanjutnya, jika disebut aljabar *Boolean*, maka aljabar *Boolean* yang dimaksudkan di sini adalah aljabar *Boolean* dua-nilai.

c. Ekspresi *Boolean*

Pada aljabar *Boolean* dua-nilai, $B = \{0, 1\}$. Kedua elemen *B* ini seringkali disebut elemen biner atau *bit* (singkatan *binary bit*). Peubah (*variable*) *x* disebut peubah *Boolean* atau peubah biner jika nilainya hanya dari *B*. Ekspresi *Boolean* dibentuk dari elemen – elemen *B* dan / atau peubah – peubah yang dapat dikombinasikan satu sama lain dengan operator +, ., dan ‘.

Dalam penulisan ekspresi *Boolean* selanjutnya, kita menggunakan perjanjian berikut : tanda kurung ‘()’ mempunyai prioritas pengerjaan paling tinggi, kemudian diikuti dengan operator ‘, + dan A. Sebagai contoh, ekspresi $a + b . c$ berarti $a + (b . c)$, bukan $(a + b) . c$ dan ekspresi $a . b'$ berarti $a . (b')$, bukan $(a . b)'$.D

d. **Prinsip Dualitas**

Di dalam aljabar *boolean*, banyak ditemukan kesamaan (*identity*) yang dapat diperoleh dari kesamaan lainnya, misalnya pada dua aksioma distributif yang sudah disebutkan pada definisi 2.1 :

(i) $a . (b + c) = (a . b) + (a . c)$
(ii) $a + (b . c) = (a + b) . (a + c)$

Aksioma yang kedua diperoleh dari aksioma pertama dengan cara mengganti A dengan + dan mengganti + dengan A. Prinsip ini dikenal dengan prinsip dualitas, prinsip yang juga kita temukan di dalam teori himpunan maupun logika. Definisi prinsip dualitas di dalam aljabar *Boolean* adalah sebagai berikut.

(Definisi 7.3) Misalkan *S* adalah kesamaan (*identity*) di dalam aljabar *Boolean* yang melibatkan operator +, A, dan ‘, maka jika pernyataan *S** diperoleh dari *S* dengan cara mengganti A dengan +, + dengan A, 0 dengan 1, 1 dengan 0 dan membiarkan operator komplemen tetap apa adanya, maka kesamaan *S** juga benar. *S** disebut sebagai *dual* dari *S*.

e. Fungsi *Boolean*

Fungsi *boolean* (disebut juga fungsi biner) adalah pemetaan dari Bn ke B melalui ekspresi *Boolean*, kita menuliskannya sebagai

$f : B^n \rightarrow B$

yang dalam hal ini B^n adalah himpunan yang beranggotakan pasangan terurut ganda-n (*ordered n-tuple*) di dalam daerah asal *B*.

Misalkan ekspresi *Boolean* dengan *n* peubah adalah E (x_1, x_2, ..., x_n). Menurut definisi di atas, setiap pemberian nilai – nilai kepada peubah x_1, x_2, ..., x_n merupakan suatu pasangan terurut ganda-*n* di dalam daerah asal B^n dan nilai ekspresi tersebut adalah bayangannya di dalam daerah hasil *B*. Dengan kata lain, setiap ekspresi *Boolean* tidak lain merupakan fungsi *boolean*. Misalkan sebuah fungsi *Boolean* adalah $f(x, y, z) = xyz + x'y + y'z$. Fungsi *f* memetakan nilai – nilai pasangan terurut ganda-3 (*x*, *y*, *z*) ke himpunan {0, 1}. Contoh pasangan terurut ganda-3 misalnya (1, 0, 1) yang berarti $x = 1$, $y = 0$, dan $z = 1$ sehingga $f(1, 0, 1) = 1 . 0 . 1 + 1' . 0 + 0' . 1 = 0 + 0 + 1 = 1$.

Selain secara aljabar, fungsi *Boolean* juga dapat dinyatakan dengan tabel kebenaran dan dengan rangkaian logika. Tabel kebenaran berisi nilai – nilai fungsi untuk semua kombinasi nilai – nilai peubahnya. Jika fungsi *Boolean* dinyatakan dengan tabel kebenaran, maka untuk fungsi *Boolean* dengan *n* buah peubah, kombinasi dari nilai peubah – peubahnya adalah sebanyak 2^n. Ini berarti terdapat 2^n baris yang berbeda di dalam tabel kebenaran tersebut. Misalkan $n = 3$, maka akan terdapat $2^3 = 8$ baris tabel. Cara yang praktis membuat semua kombinasi tersebut adalah sebagai berikut :

1. Untuk peubah pertama, isi 4 baris pertama pada kolom pertama dengan sebuah 0 dan 4 baris selanjutnya dengan sebuah 1 berturut – turut.
2. Untuk peubah kedua, isi 2 baris pertama pada kolom kedua dengan 0 dan 2 baris berikutnya dengan 1, 2 baris berikutnya 0 lagi, dan 2 baris terakhir dengan 1.
3. Untuk peubah ketiga, isi kolom ketiga secara berselang – seling dengan 0 dan 1 mulai baris pertama sampai baris terakhir.

Fungsi *Boolean* tidak selalu unik pada representasi ekspresinya. Artinya, dua buah fungsi yang ekspresi *Boolean*nya berbeda dapat menyatakan dua buah fungsi yang sama. Misalkan *f* dan *g* adalah ekspresi dari suatu fungsi *Boolean*. Fungsi *f* dan *g* dikatakan merupakan fungsi yang sama jika keduanya memiliki nilai yang sama pada tabel kebenaran untuk setiap kombinasi peubah – peubahnya. Sebagai contoh, fungsi :

$f(x, y, z) = x'y'z + x'yz + xy'$ dan $g(x, y, z) = x'z + xy'$

adalah dua buah fungsi *Boolean* yang sama. Kesamaan ini dapat dilihat pada tabel berikut.

Tabel 2.6 Tabel kebenaran fungsi f dan g

x	y	z	f = x'y'z + x'yz + xy'	g = x'z + xy'
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

Jika sebuah fungsi *Boolean* tidak unik dalam representasi ekspresinya, kita masih dapat menemukan ekspresi *Boolean* lainnya yang menspesifikasikan fungsi yang sama dengan melakukan manipulasi aljabar terhadap ekspresi *Boolean*. Yang dimaksud dengan memanipulasi atau menyederhanakan fungsi *Boolean* adalah menggunakan hukum – hukum aljabar *Boolean* untuk menghasilkan bentuk yang ekivalen. Sebagai contoh,

$f(x, y, z) = x'y'z + x'yz + xy'$
$= x'z(y' + y) + xy'$ (Hukum distributif)
$= x'z . 1 + xy'$ (Hukum komplemen)
$= x'z + xy'$ (Hukum identitas)

Manipulasi aljabar pada ekspresi *Boolean* disebut juga dengan penyederhanaan fungsi *Boolean*.

f. Metode Quine-McCluskey

Metode peta Karnaugh hanya cocok digunakan jika fungsi *Boolean* mempunyai jumlah peubah yang tidak besar. Jika peubah yang terlibat pada suatu fungsi *Boolean* dalam jumlah yang besar maka penggunaan peta Karnaugh menjadi semakin rumit, sebab ukuran peta bertambah besar. Selain itu, metode peta Karnaugh lebih sulit diprogram dengan komputer karena diperlukan pengamatan visual untuk mengidentifikasi *minterm – minterm* yang akan dikelompokkan. Untuk itu diperlukan metode penyederhanaan yang lain yang dapat diprogram dan dapat digunakan untuk fungsi *Boolean* dengan sembarang jumlah peubah. Metode alternatif tersebut adalah metode Quine-McCluskey.

Metode Quine-McCluskey adalah sebuah metode yang digunakan untuk menyederhanakan fungsi *Boolean*, khususnya fungsi *Boolean* yang memiliki jumlah peubah yang besar (di atas 6 buah). Metode Quine-McCluskey dikembangkan oleh W.V. Quine dan E.J. McCluskey pada tahun 1950.

Metode ini mengubah sebuah fungsi *Boolean* menjadi sebuah himpunan bentuk prima, dimana sebanyak mungkin peubah dieliminasi (dihilangkan) secara maksimal, hingga didapat fungsi *Boolean* yang paling sederhana. Ini dapat dilakukan dengan melakukan perulangan penggunaan hukum komplemen, $a + a' = 1$. Sebagai contoh, fungsi *Boolean* dengan empat peubah dalam bentuk SOP : $f(a, b, c, d) = 3(3, 11) = 3(0011, 1011) = a'b'cd + ab'cd$ dan $f(a, b, c, d) = 3(7, 11) = 3(0111, 1011) = a'b'cd + ab'cd$.

	abcd
3	0 0 1 1
11	1 0 1 1
BENTUK PRIMA -> (3,11)	- 0 1 1

Contoh (a)

	abcd
7	0 1 1 1
11	1 0 1 1
?	

Contoh (b)

Pada contoh(a), kedua *minterm* tersebut dapat dikombinasikan menjadi sebuah bentuk prima yaitu (3,11), karena memiliki tepat satu perbedaan bit pada posisi bit nomor satu. Hasil kombinasi dalam bentuk prima (3,11) menyatakan bahwa peubah '*a*' telah dieleminasi. Hal ini sesuai dengan hukum komplemen, $a + a' = 1$.

Pada contoh(b), kedua *minterm* tersebut tidak dapat dikombinasikan menjadi sebuah bentuk prima, karena memiliki dua perbedaan bit pada posisi bit nomor satu dan dua. Setiap kombinasi dari *minterm* yang dapat membentuk sebuah bentuk prima baru harus memiliki tepat satu perbedaan bit pada posisi yang sama.

Secara umum, langkah – langkah metode Quine-McCluskey untuk menyederhanakan ekspresi *Boolean* dalam bentuk SOP adalah sebagai berikut :

1. Nyatakan tiap *minterm* dalam *n* peubah menjadi *string bit* yang panjangnya *n*, yang dalam hal ini peubah komplemen dinyatakan dengan '0', peubah yang bukan komplemen dengan '1'.
2. Kelompokkan tiap *minterm* berdasarkan jumlah '1' yang dimilikinya.
3. Kombinasikan *minterm* dalam *n* peubah dengan kelompok lain yang jumlah '1'-nya berbeda satu, sehingga diperoleh bentuk prima (*prime-implicant*) yang terdiri dari *n* - 1 peubah. *Minterm* yang dikombinasikan diberi tanda "√".
4. Kombinasikan *minterm* dalam *n* – 1 peubah dengan kelompok lain yang jumlah '1'-nya berbeda satu, sehingga diperoleh bentuk prima yang terdiri dari *n* – 2 peubah.
5. Teruskan langkah 4 sampai diperoleh bentuk prima yang sesederhanan mungkin.
6. Ambil semua bentuk prima yang tidak bertanda "√". Buatlah tabel baru yang memperlihatkan *minterm* dari ekspresi *Boolean* semula yang dicakup oleh bentuk prima tersebut (tandai dengan "×"). Setiap *minterm* harus dicakup oleh paling sedikit satu buah bentuk prima.
7. Pilih bentuk prima yang memiliki jumlah literal paling sedikit namun mencakup sebanyak mungkin *minterm* dari ekspresi *Boolean* semula. Hal ini dapat dilakukan dengan cara berikut :
   a. Tandai kolom – kolom yang mempunyai satu buah tanda "×" dengan tanda "*", lalu beri tanda "√" di sebelah kiri bentuk prima yang berasosiasi dengan tanda "*" tersebut. Bentuk prima ini telah dipilih untuk fungsi *Boolean* sederhana.
   b. Untuk setiap bentuk prima yang telah ditandai dengan "√", beri tanda *minterm* yang dicakup oleh bentuk prima tersebut dengan tanda "√" (di baris bawah setelah "*").
   c. Periksa apakah masih ada *minterm* yang belum dicakup oleh bentuk prima terpilih. Jika ada, pilih dari bentuk prima yang tersisa yang mencakup sebanyak mungkin *minterm* tersebut. Beri tanda "√" bentuk prima yang dipilih itu serta *minterm* yang dicakupnya.
   d. Ulangi langkah c sampai seluruh *minterm* sudah dicakup oleh semua bentuk prima.

Langkah – langkah penyederhanaan metode Quine-McCluskey di atas juga berlaku untuk penyederhanaan fungsi *Boolean* dalam bentuk POS. Perhatikan bahwa bentuk fungsi *output* selalu sama dengan bentuk fungsi *input*, artinya *input* dalam bentuk SOP akan menghasilkan *output* dalam bentuk dalam SOP, dan demikian pula untuk bentuk POS.

III. Pembahasan

Metode Quine-McCluskey menyederhanakan fungsi *boolean* dengan menggabungkan *minterm / maxterm* menjadi himpunan bentuk prima (*prime-implicant*), dimana sebanyak mungkin peubah dieliminasi (dihilangkan) secara maksimal. *Minterm / maxterm* yang digabung untuk membentuk sebuah bentuk prima harus

memiliki tepat satu buah peubah yang nilainya berbeda (komplemen dan non-komplemen). Bentuk prima yang terbentuk juga dapat digabung untuk membentuk bentuk prima lainnya, apabila memiliki satu buah peubah yang nilainya berbeda. Prosedur ini dilakukan berulang kali hingga tidak terdapat bentuk prima yang dapat terbentuk lagi.

Untuk proses selanjutnya, metode ini memilih bentuk prima yang paling sederhana, yaitu bentuk prima yang tidak membentuk bentuk prima yang baru. Dari bentuk-bentuk prima yang sederhana ini, dipilih bentuk prima yang memiliki jumlah peubah paling sedikit (bentuk prima yang paling panjang) dan mencakup sebanyak mungkin *minterm / maxterm* dari fungsi *boolean* yang di-*input*. Bentuk prima inilah yang merupakan hasil minimisasi dari fungsi *boolean*.

Keunggulan dari metode Quine-McCluskey adalah kemampuannya untuk melakukan proses penyederhanaan terhadap fungsi *boolean* dengan jumlah peubah yang besar. Selain itu, prosedur atau langkah – langkah pengerjaan dari metode ini sangat sistematis, artinya pengerjaan setiap langkah sangat rapi, teratur dan terstruktur, sehingga metode ini lebih mudah diprogram di dalam komputer dibandingkan dengan metode penyederhanaan lainnya.

Sebagai contoh, dilakukan penyederhanaan terhadap fungsi *boolean* dalam bentuk SOP : $f(w, x, y, z) =$ (0, 1, 3, 4, 5, 6, 8, 10, 11, 15). Langkah – langkah penyederhanaan fungsi Boolean dengan metode Quine-McCluskey adalah sebagai berikut:

LANGKAH-1 : Nyatakan tiap *minterm* dalam n peubah menjadi string bit biner yang panjangnya n. (Pada contoh ini, jumlah peubah adalah 4 sehingga n = 4)

0 = 0000
1 = 0001
3 = 0011
4 = 0100
5 = 0101
6 = 0110
8 = 1000
10 = 1010
11 = 1011
15 = 1111

LANGKAH-2 : Kelompokkan tiap *minterm* berdasarkan jumlah '1' yang dimilikinya.

Hasil Penyelesaian Langkah - 2 :

```
))))))))))))))))
 term   w x y z
))))))))))))))))
 0      0 0 0 0  -> jumlah bit '1' = 0 buah
))))))))))))))))
 1      0 0 0 1  -> jumlah bit '1' = 1 buah
 4      0 1 0 0
 8      1 0 0 0
))))))))))))))))
 5      0 1 0 1  -> jumlah bit '1' = 2 buah
 6      0 1 1 0
 3      0 0 1 1
 10     1 0 1 0
))))))))))))))))
 11     1 0 1 1  -> jumlah bit '1' = 3 buah
))))))))))))))))
 15     1 1 1 1  -> jumlah bit '1' = 4 buah
))))))))))))))))
```

LANGKAH-3 : Kombinasikan *minterm* dalam n peubah dengan kelompok lain yang jumlah '1'-nya berbeda satu, sehingga diperoleh bentuk prima (*prime-implicant*) yang terdiri dari n-1 peubah. *minterm* yang dikombinasikan diberi tanda 'v'

LANGKAH-4 : Kombinasikan *minterm* dalam n - 1 peubah dengan kelompok lain yang jumlah '1'-nya berbeda satu, sehingga diperoleh bentuk prima yang terdiri dari n-2 peubah.

LANGKAH-5 : Teruskan langkah 4 sampai diperoleh bentuk prima yang sesederhana mungkin.

Hasil Penyelesaian Langkah-3, 4 dan 5 :

```
))))))))))))))))
 term   w x y z
))))))))))))))))
 0      0 0 0 0   v
))))))))))))))))
 1      0 0 0 1   v
 4      0 1 0 0   v
 8      1 0 0 0   v
))))))))))))))))
 5      0 1 0 1   v
 6      0 1 1 0   v
 3      0 0 1 1   v
 10     1 0 1 0   v
))))))))))))))))
 11     1 0 1 1   v
))))))))))))))))
 15     1 1 1 1   v
))))))))))))))))
```

Dikombinasikan menjadi :

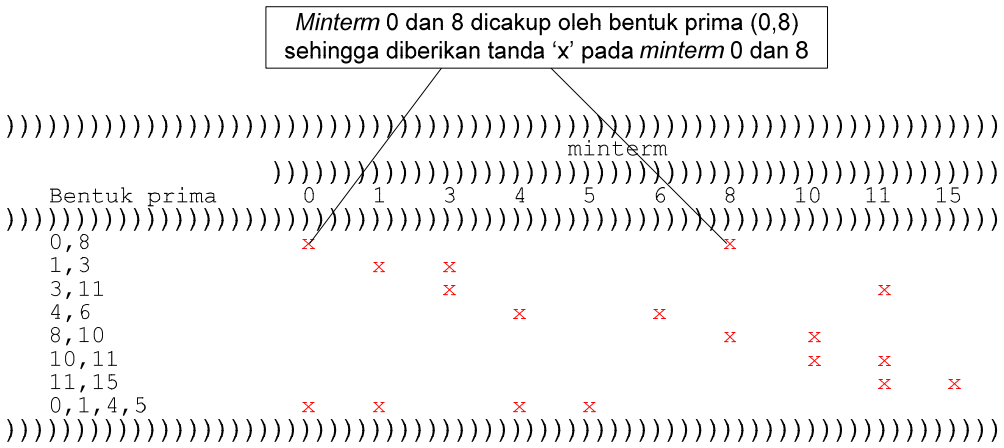
```
))))))))))))))))
 term   w x y z
))))))))))))))))
 0,1    0 0 0 -   v
 0,4    0 - 0 0   v
 0,8    - 0 0 0
))))))))))))))))
 1,3    0 0 - 1
 1,5    0 - 0 1   v
 4,5    0 1 0 -   v
 4,6    0 1 - 0
 8,10   1 0 - 0
))))))))))))))))
 3,11   - 0 1 1
 10,11  1 0 1 -
))))))))))))))))
 11,15  1 - 1 1
))))))))))))))))
```

Dikombinasikan menjadi :

```
))))))))))))))))))))
 term       w x y z
))))))))))))))))))))
 0,1,4,5    0 - 0 -
 0,4,1,5    0 - 0 -
))))))))))))))))))))
```

LANGKAH-6 : Ambil semua bentuk prima yang tidak bertanda 'v'. Buatlah tabel baru yang memperlihatkan *minterm* dari ekspresi Boolean semula yang dicakup oleh bentuk prima tersebut (tandai dengan 'x').

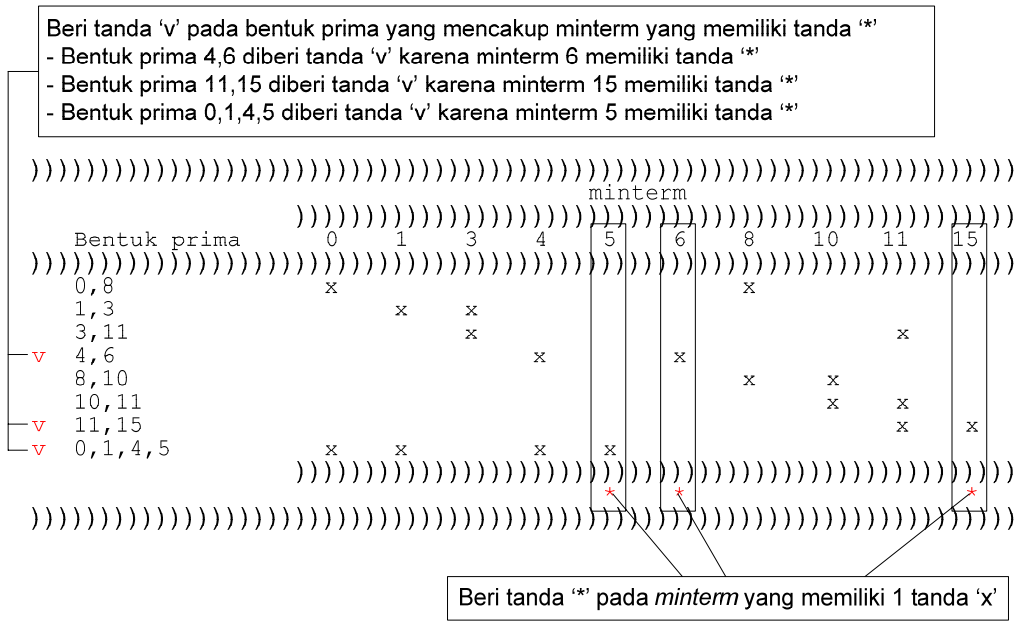
Hasil Penyelesaian Langkah - 6 :



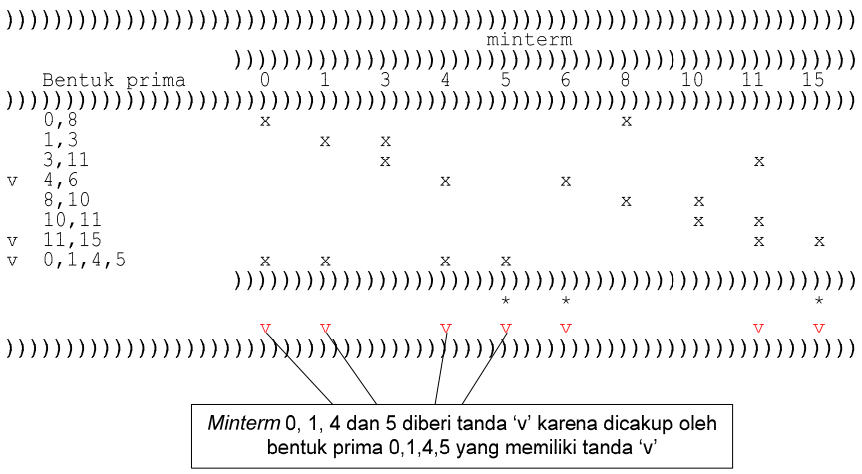
LANGKAH-7 : Pilih bentuk prima yang memiliki jumlah literal paling sedikit namun mencakup sebanyak mungkin minterm dari ekspresi Boolean semula. Hal ini dapat dilakukan dengan cara berikut :

LANGKAH-7.A : Tandai kolom-kolom yang mempunyai satu buah tanda 'x' dengan tanda '*', lalu beri tanda 'v' di sebelah kiri bentuk prima yang mencakup minterm yang mempunyai tanda '*' tersebut. Bentuk prima ini telah dipilih untuk fungsi Boolean sederhana.

Hasil Penyelesaian Langkah-7 dan 7.A :

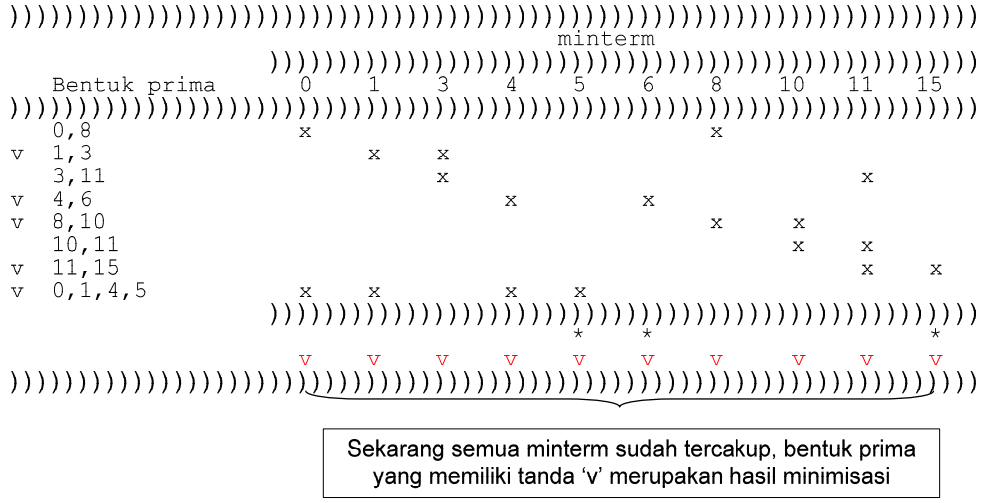


LANGKAH-7.B : Untuk setiap bentuk prima yang telah ditandai dengan 'v', beri tanda *minterm* yang dicakup oleh bentuk prima tersebut dengan tanda 'v' (di baris bawah setelah tanda '*').
Hasil Penyelesaian Langkah-7.B :



LANGKAH-7.C : Periksa apakah masih ada minterm yang belum memiliki tanda 'v' (artinya, belum dicakup oleh bentuk prima terpilih). Jika ada, pilih dari bentuk prima yang tersisa yang mencakup sebanyak mungkin minterm tersebut. Beri tanda 'v' untuk setiap bentuk prima yang dipilih itu serta minterm yang dicakupnya.

LANGKAH-7.D : Ulangi langkah 7.C sampai seluruh minterm sudah dicakup oleh bentuk prima
Hasil Penyelesaian Langkah - 7.C dan 7.D :
Sampai tahap ini, masih ada *minterm* yang belum tercakup dalam bentuk prima terpilih, yaitu 3, 8, 10. Untuk mencakup *minterm* tersebut, kita pilih bentuk prima (8,10) dan (1,3) karena mencakup *minterm* 3, 8 dan 10 sekaligus.



Sekarang, semua minterm sudah tercakup dalam bentuk prima terpilih. Bentuk prima yang terpilih adalah :
1, 3 yang bersesuaian dengan *termw'x'z* (*minterm* 1 (0001) dan *minterm* 3 (0011) memiliki persamaan bit pada posisi 1, 2 dan 4 sehingga dapat dikatakan *minterm* 1 dan *minterm* 3 merupakan fungsi Boolean *w'x'z*).
4, 6 yang bersesuaian dengan *termw'xz'*
8, 10 yang bersesuaian dengan *termwx'z'*
11, 15 yang bersesuaian dengan *termwyz*
0, 1, 4, 5 yang bersesuaian dengan *termw'y'*
Dari langkah – langkah penyederhanaan di atas, hasil minimisasi fungsi Boolean dalam bentuk SOP: $f(w, x, y, z) = 3$ (0, 1, 3, 4, 5, 6, 8, 10, 11, 15) adalah
$f(w, x, y, z) = w'x'z + w'xz' + wx'z' + wyz + w'y'$.

Misalkan *input* fungsi dalam bentuk POS dengan nomor – nomor *maxterm* yang sama dengan nomor – nomor *minterm* pada contoh sebelumnya.

$f(w, x, y, z) = \vartheta\ (0, 1, 3, 4, 5, 6, 8, 10, 11, 15)$

Langkah – langkah dan hasil penyederhanaan sama dengan penyederhanaan dalam bentuk SOP, yaitu menghasilkan bentuk prima terpilih (1,3), (4,6), (8,10), (11,15) dan (0,1,4,5). Perbedaannya hanya terletak pada bentuk *output*. Bentuk prima terpilih diubah ke bentuk POS (sama dengan bentuk *input*).

1, 3 bersesuaian dengan *term* $(w + x + z')$.
4, 6 yang bersesuaian dengan *term* $(w + x' + z)$.
8, 10 yang bersesuaian dengan *term* $(w' + x + z)$.
11, 15 yang bersesuaian dengan *term* $(w' + y' + z')$.
0, 1, 4, 5 yang bersesuaian dengan *term* $(w + y)$.

Dengan demikian, hasil minimisasi fungsi Boolean dalam bentuk POS adalah

$f(w, x, y, z) = (w + x + z')(w + x' + z)(w' + x + z)(w' + y' + z')(w + y)$.

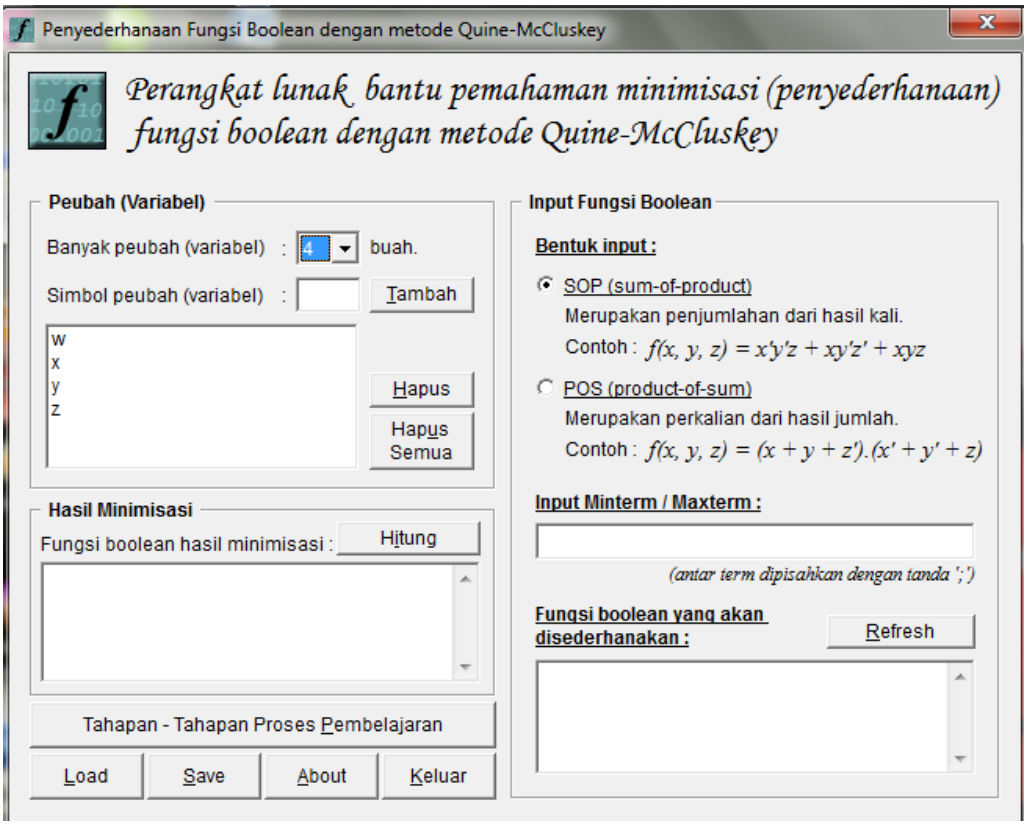
IV. Analsis Hasil

Perangkat lunak bantu pemahaman ini dirancang dengan menggunakan bahasa pemrograman *Microsoft Visual Basic 6.0* dengan beberapa komponen standar seperti *Command Button, Text Box, Rich Text Box, Option Button, Label, Shape*, dan sebagainya.

Perangkat lunak bantu pemahaman ini memiliki beberapa *form*, antara lain :

Form Input

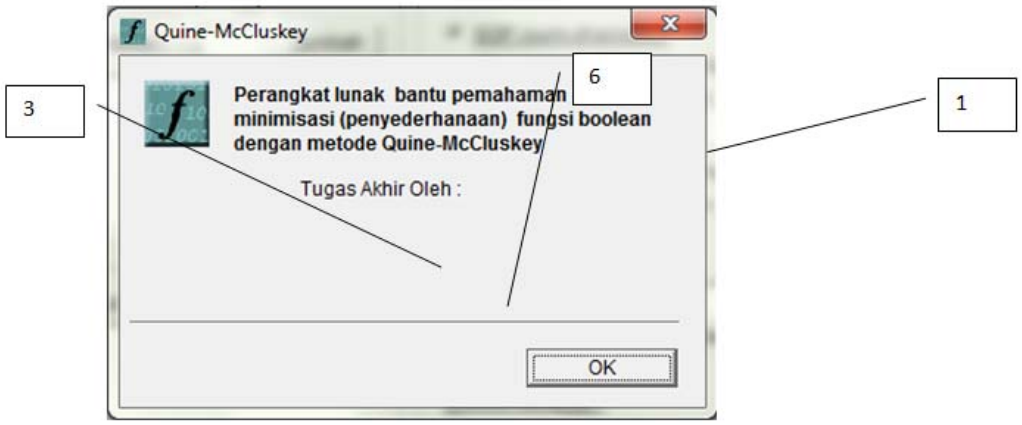
FormInput berfungsi sebagai *form* untuk memasukkan *input* terhadap perangkat lunak, seperti: banyak peubah, simbol peubah, bentuk fungsi dan *inputterm* pada fungsi Boolean. *Input* fungsi Boolean dapat disimpan (*save*) dalam bentuk *file* (*exstension* QMC) dan dibuka (*load*) kembali. Pada *form* ini, juga terdapat fasilitas untuk menghitung dan mendapatkan hasil minimisasi fungsi Boolean secara langsung (tanpa melihat langkah – langkah pemahaman).



Gambar 3.2.1 Rancangan Form Input

3.2.2 Form About

FormAbout berfungsi sebagai *form* untuk menampilkan informasi mengenai pembuat perangkat lunak



Gambar 3.2.2 Rancangan Form About

4.1.1 Algoritma Penghasil Tahapan – Tahapan Penyederhanaan Fungsi *Boolean* dengan metode Quine-McCluskey

Algoritma ini merupakan inti dari perangkat lunak. Fungsi dari algoritma ini adalah melakukan proses penyederhanaan terhadap *input* fungsi *boolean* dengan metode Quine-McCluskey dan sekaligus menghasilkan tahapan – tahapan penyederhanaan dalam format *text*. Tahapan – tahapan penyederhanaan ini dimaksudkan sebagai fasilitas pembelajaran bagi *user*. Tahapan penyederhanaan ini juga dapat disimpan (format *rich text file* (*.rtf)), dibuka dengan aplikasi *Word Processing* seperti *Microsoft Word* atau *WordPad*, dan dicetak ke *printer*. Algoritma ini dirancang dalam bentuk fungsi dengan nama 'Proses QuineMcCluskey'. Fungsi ini menghasilkan tahapan – tahapan penyederhanaan, menyimpannya ke variabel *array* cLangkah dan mengembalikan hasil minimisasi. Algoritma dari fungsi 'ProsesQuineMcCluskey' adalah sebagai berikut,

1. Untuk mengerjakan langkah-1 dari metode Quine McCluskey, lakukan langkah berikut. Untuk N1 = 0 sampai jumlah *array* InputTerm.
   a. Set MTerm(N1).Term = InputTerm(N1), set MTerm(N1).Bit = bilangan biner dari InputTerm(N1) dengan panjang bilangan sebesar banyak_peubah, set MTerm(N1).Level = 1, set MTerm(N1).Kelompok = banyak jumlah bit '1' dari bilangan biner, set MTerm(N1).Check = *False*.
   b. Set Pos(N1) = N1.
   c. Untuk langkah penyelesaian, set cLangkah(1) = cLangkah(1) & vbCrlf & MTerm(N1).Term & " = " & MTerm(N1).Bit.
2. Untuk mengerjakan langkah-2 sampai langkah-5 dari metode Quine McCluskey, set nLevel1 = 1. Selama bSederhana = *False* (masih bisa dikelompokkan atau belum sederhana), lakukan langkah berikut.
   a. Set nLevel = nLevel + 1.
   b. Periksa apakah terdapat *term* yang memiliki perbedaan bit sebanyak 1 buah untuk *level* yang sama. Apabila ada, maka bentuk sebuah bentuk prima yang baru. Untuk N1 = 0 sampai jumlah *array* dari MTerm, lakukan prosedur berikut.
      i. Jika MTerm(N1).Level = nLevel maka lakukan *looping* dari N2 = N1 + 1 sampai jumlah array MTerm, cek jika MTerm(N2).Level = nLevel maka lakukan langkah berikut.
         1. Ambil banyak perbedaan bit, dengan men-set DF = GetDifference(MTerm(N1).Bit, MTerm(N2).Bit).
         2. Jika DF.Banyak = 1(jumlah bit yang berbeda sebanyak 1 buah), maka lanjutkan ke poin 3.
         3. Set MTerm(N1).Check = *True*, set MTerm(N2).Check = *True*.
         4. Set N3 = jumlah *array* MTerm + 1. Bentuk ulang dimensi *array* dari MTerm menjadi N3.
         5. Set MTerm(N3).Term = MTerm(N1).Term & "," & MTerm(N2).Term. Ini adalah bentuk prima yang baru.
         6. MTerm(N3).Bit = Left(MTerm(N1).Bit, DF.Posisi – 1) & "-" & Mid(MTerm(N1).Bit, DF.Posisi + 1, Len(MTerm(N1).Bit) – DF.Posisi).
         7. Set MTerm(N3).Check = False
         8. Set MTerm(N3).Level = nLevel + 1.
         9. Set MTerm(N3).Kelompok = MTerm(N1).Kelompok.
         10. Set nLevel1 = nLevel + 1

c. Set bSederhana = *True*.
d. Cek apakah masih bisa disederhanakan lagi. Untuk N1 = 0 sampai jumlah *array* dari MTerm, lakukan langkah berikut.
   i. Jika MTerm(N1).Level = nLevel + 1, maka lakukan *looping* dari N2 = N1 + 1 sampai jumlah *array* MTerm, lakukan langkah berikut.
      1. Jika MTerm(N2).Level = nLevel + 1, maka set DF = GetDifference(MTerm(N1).Bit, MTerm(N2).Bit).
      2. Jika DF.Banyak = 1 maka set bSederhana = False dan keluar dari *looping*.
   ii. Jika bSederhana = False, maka keluar dari *looping*.

3. Ambil semua bentuk prima yang tidak memiliki tanda *check* dan *update* ke bentuk prima (Langkah 6.1 dari metode Quine McCluskey). Algoritmanya adalah sebagai berikut,
   a. Set N5 = 0.
   b. Bentuk ulang dimensi *array* variabel Prima menjadi 0.
   c. Untuk N1 = 0 sampai jumlah *array* dari MTerm, lakukan langkah berikut,
      i. Jika MTerm(N1).Check = *False* maka lakukan langkah berikut,
         1. Urutkan *term* dengan men-set cTemp = Urutkan (MTerm (N1).Term)
         2. Cek apakah bentuk prima sudah pernah ada dengan set bSama = False, lakukan *looping* dari N2 = 1 sampai jumlah *array* dari variabel Prima, jika Prima(N2).Term = cTemp maka set bSama = *True* dan keluar dari *looping*.
         3. Jika bSama = False, maka lakukan langkah berikut,
            a. Set N3 = Ubound(Prima) + 1.
            b. Bentuk ulang dimensi *array* Prima menjadi N3.
            c. Set Prima(N3).Term = cTemp.
            d. Set Prima(N3).Check = False.
            e. Set Prima(N3).Nilai = 0.
            f. Jika panjang dari Prima(N3).Term > N5 maka, set N5 = panjang dari Prima(N3).Term.

4. Beri tanda ‘x’ untuk setiap *term* yang tercakup di dalam bentuk prima yang terpilih. Algoritmanya adalah sebagai berikut,
   a. Bentuk ulang dimensi *array* variabel MTermP menjadi jumlah *array* dari variabel InputTerm.
   b. Untuk N1 = 0 sampai jumlah *array* variabel InputTerm, lakukan langkah berikut,
      i. Set MTermP(N1).Term = InputTerm(N1).
      ii. Set MTermP(N1).BanyakX = 0.
      iii. Set MTermP(N1).Check = *False*.
      iv. Set MTermP(N1).Bintang = *False*.
   c. Untuk N1 = 1 sampai jumlah *array* variabel Prima, lakukan langkah berikut,

Untuk N2 = 0 sampai jumlah *array* variabel MTermP, cek jika MTermP(N2).Term terdapat di dalam list dari Prima(N1).Term) maka set MTermP(N2).BanyakX = MTermP(N2).BanyakX + 1.

5. Tandai ‘*’ untuk semua MTermP yang memiliki 1 buah tanda ‘x’ (Langkah 7.a.1 dari metode Quine McCluskey). Algoritmanya adalah sebagai berikut,
   a. Untuk N1 = 0 sampai jumlah *array* dari variabel MTermP, cek jika MTermP(N1).BanyakX = 1 maka set MTermP(N1).Bintang = *True*.
6. Beri tanda *check* pada semua bentuk prima yang mencantumkan *term* yang memiliki tanda ‘*’ (Langkah 7.a.2 dari metode Quine McCluskey). Algoritmanya adalah sebagai berikut,
   a. Untuk N1 = 0 sampai jumlah *array* dari variabel MTermP, lakukan langkah berikut,

Jika MTermP(N1).Bintang = *True*, maka lakukan *looping* dari N2 = 1 sampai jumlah *array* dari variabel Prima, cek jika MTermP(N1).Term tercantum di dalam Prima(N2).Term, maka set Prima(N2).Check = *True*.

7. Beri tanda *check* di MTermP untuk semua MTermP yang tercantum dalam bentuk prima yang memiliki tanda *check* (Langkah 7.b dari metode Quine McCluskey). Algoritmanya adalah sebagai berikut,
   a. Untuk N1 = 0 sampai jumlah *array* dari variabel Prima, lakukan langkah berikut,
      Jika Prima(N1).Check = *True*, maka lakukan looping dari N2 = 0 sampai jumlah array variabel MTermP, cek jika MTermP(N2).Term tercantum di dalam Prima(N1).Term, maka set MTermP(N2).Check = *True*.
8. Periksa apakah masih ada MTermP yang belum ter-*check*, apabila ada, maka pilih bentuk prima yang mencakup paling banyak MTermP yang belum ter-*check* (Langkah 7.c & 7.d dari metode Quine McCluskey). Algoritmanya adalah sebagai berikut,
   a. Set bAllCheck = FAllCheck(MTermP).
   b. Jika bAllCheck = *False*, maka lanjutkan algoritma pada poin c.
   c. Set cTemp = “”.
   d. Untuk N1 = 0 sampai jumlah *array* dari variabel MTermP, lakukan langkah berikut,

Jika MTermP(N1).Check = *False*, maka
1. Jika cTemp <> “” maka cTemp = cTemp & “, “.
2. Set cTemp = cTemp & MTermP(N1).Term.
3. Untuk N2 = 1 sampai jumlah *array* dari variabel Prima, cek jika MTermP(N1).Term tercantum di dalam Prima(N2).Term maka Prima(N2).Nilai = Prima(N2).Nilai + 1.

e. Bentuk ulang dimensi *array* variabel Pos menjadi jumlah *array* dari variabel Prima.
f. Untuk N1 = 1 sampai jumlah *array* dari variabel Pos, set Pos(N1) = N1.
g. Kemudian, lakukan pengurutan variabel Prima berdasarkan nilai yang dimiliki. Pengurutan yang dilakukan menggunakan metode *buble sort*.
   i. Untuk N1 = 1 sampai jumlah *array* dari variabel Pos – 1, lakukan langkah berikut.
   ii. Untuk N2 = N1 + 1 sampai jumlah array dari variabel Pos, lakukan langkah berikut.
   iii. Jika Prima(Pos(N1)).Nilai < Prima(Pos(N2)).Nilai, maka
      1. Set N3 = Pos(N1).
      2. Set Pos(N1) = Pos(N2).
      3. Set Pos(N2) = N3.
   iv. Jika Prima(Pos(N1)).Nilai = Prima(Pos(N2)).Nilai, maka pilih bentuk prima yang paling panjang.
      1. Cek jika UBound(Split(Prima(Pos(N1)).Term, ",")) < UBound(Split(Prima(Pos(N2)).Term, ",")), maka tukar posisi.
         a. Set N3 = Pos(N1).
         b. Set Pos(N1) = Pos(N2).
         c. Set Pos(N2) = N3.
h. Set cPrima = “”.
i. Set N1 = 0.
j. Selama bAllCheck = *False*, lakukan langkah berikut,
   i. Set N1 = N1 + 1.
   ii. Set Prima(Pos(N1)).Check = *True*.
   iii. Jika cPrima <> “” maka set cPrima = cPrima & “, “.
   iv. Untuk N2 = 0 sampai jumlah *array* dari variabel MTermP, lakukan langkah berikut,
      1. Jika MTermP(N2).Check = False, maka cek jika MTermP(N2).Term tercantum dalam Prima(Pos(N1)).Term) maka set MTermP(N2).Check = *True*.
   v. Set bAllCheck = FAllCheck(MTermP).

9. Set HasilMinimisasi = “”.
10. Ambil bentuk prima yang ter-*check* dan ubah menjadi bentuk kanonik. Algoritmanya adalah sebagai berikut,
    a. Untuk N1 = 1 sampai jumlah *array* dari variabel Prima, lakukan langkah pada poin b berikut.
    b. Jika Prima(N1).Check maka,
       i. Set Prima(N1).Bentuk Kanonik = PrimaToKanonik(Prima(N1).Term).
       ii. Jika cBentuk = “SOP”, maka,
          1. Jika HasilMinimisasi <> “” maka set Hasil Minimisasi = Hasil Minimisasi & “ + “.
          2. Set Hasil Minimisasi = Hasil Minimisasi & Prima(N1).Bentuk Kanonik.
       iii. Jika cBentuk = “POS”, maka,
          1. Set Hasil Minimisasi = Hasil Minimisasi & Prima(N1). Bentuk Kanonik.
11. Kembalikan hasil minimisasi.
    a. Set ProsesQuineMcCluskey = HasilMinimisasi.

DAFTAR PUSTAKA

InformatikaBandung. (2004). Pengantar Logika Matematika. Bandung: Informatika Bandung, 34.

Kenneth H. Rosen. (1994). *Discrete Mathematics andits Applications.* New York: McGraw-Hill, 120.

Kusrini. 2007. tuntunan praktis membangun sistem informasi akuntansi dengan *visual basic* dan *microsoft sql server*. Jogyakarta: Andi, 45.

Lipschutz, Seymour & Marc Lars Lipson. 1992. 2000 *Solved Problems in Discrete Mathematics.* NewYork: McGraw-Hill, 340.

Retno Hendrawati,IR,MT & BambangHariyanto.2000.LogikaMatematika. Bandung:Informatika Bandung, 15.

Rinaldi Munir. 2005. Matematika Diskrit, Bandung:Informatika Bandung, 34.