BAB II

LANDASAN TEORI

2.1 Definisi Aljabar Boolean

Aljabar Boolean dapat didefinisikan secara abstrak dalam beberapa cara. Cara yang paling umum adalah dengan menspesifikasikan unsur – unsur pembentuknya dan operasi – operasi yang menyertainya.

(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 $a, b, c \in B$ 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 Ø 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.

Terdapat perbedaan antara aljabar Boolean dengan aljabar biasa untuk aritmetika bilangan riil :

1. Hukum distributif yang pertama, $a \,.\, (b + c) = (a \,.\, b) + (a \,.\, c)$ sudah dikenal di dalam aljabar biasa, tetapi hukum distributif yang kedua, $a + (b \,.\, c) = (a + b) \,.\, (a + c)$, benar untuk aljabar Boolean, tetapi tidak benar untuk aljabar biasa.
2. Aljabar Boolean tidak memiliki kebalikan perkalian (*multiplicative inverse*) dan kebalikan penjumlahan; karena itu, tidak ada operasi pembagian dan pengurangan di dalam aljabar Boolean.
3. Aksioma nomor 4 pada definisi 2.1 mendefinisikan operator yang dinamakan *komplemen* yang tidak tersedia pada aljabar biasa.

4. Aljabar biasa memperlakukan himpunan bilangan riil dengan elemen yang tidak berhingga banyaknya. Sedangkan aljabar Boolean memperlakukan himpunan elemen *B* yang sampai sekarang belum didefinisikan, tetapi pada aljabar Boolean dua-nilai, *B* didefinisikan sebagai himpunan dengan hanya dua nilai, 0 dan 1.

Hal lain yang penting adalah membedakan elemen himpunan dan peubah (*variable*) pada sistem aljabar. Sebagai contoh, pada aljabar biasa, elemen himpunan bilangan riil adalah angka, sedangkan peubahnya seperti *a*, *b*, *c* dan sebagainya. Dengan cara yang sama pada aljabar Boolean, orang mendefinisikan elemen – elemen himpunan dan peubah seperti *x, y, z* sebagai simbol – simbol yang merepresentasikan elemen.

Berhubung elemen – elemen *B* tidak didefinisikan nilainya (kita bebas menentukan anggota – anggota *B*), maka untuk mempunyai sebuah aljabar Boolean, orang harus memperlihatkan :

1. elemen – elemen himpuan *B*,
2. kaidah / aturan operasi untuk dua operator biner dan operator uner,
3. himpunan *B*, bersama – sama dengan dua operator tersebut, memenuhi keempat aksioma.

Jika ketiga persyaratan dipenuhi, maka aljabar yang didefinisikan dapat dikatakan sebagai aljabar Boolean.

2.2 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, ('). [3] 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

Harus diperhatikan 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 dilihat 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 2.4). 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

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) $a \cdot a = 0$, karena $0 \cdot 0' = 0 \cdot 1$ dan $1 \cdot 1' = 1 \cdot 0 = 0$

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.

2.3 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 ('). Secara formal, ekspresi Boolean dapat didefinisikan secara rekursif sebagai berikut.

(Definisi 2.2 – Liu, C. L, *Element of Doscrete Mathematics, 4th, Pentice Hall,1997*) Misalkan (B, +, ., ', 0, 1) adalah sebuah aljabar Boolean. Suatu ekspresi Boolean dalam (B, +, ., ') adalah :

(i) Setiap elemen di dalam B,
(ii) setiap peubah,
(iii) jika e_1 dan e_2 adalah ekspresi Boolean, maka $e_1 + e_2$, $e_1 \cdot e_2$, e_1' adalah ekspresi Boolean.

Jadi menurut definisi 2.2, setiap ekspresi berikut,

0

1

a

b

c

$a + b$

$a \,.\, b$

$a' \,.\, (b + c)$

$a \,.\, b' + a \,.\, b \,.\, c + b'$, dan sebagainya

adalah ekspresi Boolean. Ekspresi Boolean yang mengandung n peubah dinamakan ekspresi Boolean bagi n peubah.

Dalam penulisan ekspresi Boolean selanjutnya, dapat digunakan perjanjian berikut : selai tanda kurung ‘( )’, operator (') mempunyai prioritas pengerjaan lebih tinggi daripada operator (+) dan (.) jadi sebagai contoh,

$a + b \,.\, c$ berarti $a + (b \,.\, c)$, bukan $(a + b) \,.\, c$

$a \,.\, b'$ berarti $a \,.\, (b')$, bukan (a . b)'.

2.4 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 (.) dengan (+) dan mengganti (+) dengan (.). Prinsip ini dikenal

dengan prinsip dualitas, prinsip yang juga ditemukan alam teori himpunan maupun logika. Definisi prinsip dualitas di dalam aljabar Boolean adalah sebagai berikut. [3]

> (Definisi 2.3) Misalkan *S* adalah kesamaan (*identity*) di dalam aljabar Boolean yang melibatkan operator (+), (.), dan ('), maka jika pernyataan *S** diperoleh dari *S* dengan cara mengganti
> (.) dengan (+)
> (+) dengan (.)
> (0) dengan (1)
> (1) dengan (0)
> dan membiarkan operator komplemen tetap apa adanya, maka kesamaan *S** juga benar. *S** disebut sebagai *dual* dari *S*.

2.5 Hukum – Hukum Aljabar Boolean

Ada banyak hukum di dalam aljabar Boolean. Beberapa literatur bervariasi dalam mengungkapkan jumlah hukum pada aljabar Boolean, tetapi hukum – hukum yang paling penting ditampilkan pada tabel berikut. [3]

Tabel 2.5 Tabel hukum – hukum aljabar Boolean

1. Hukum identitas : (i) $a + 0 = a$ (ii) $a \,.\, 1 = a$	2. Hukum idempoten : (i) $a + a = a$ (ii) $a \,.\, a = a$
3. Hukum komplemen : (i) $a + a' = 1$ (ii) $a \,.\, a' = 0$	4. Hukum dominansi : (i) $a \,.\, 0 = 0$ (ii) $a + 1 = 1$
5. Hukum involusi : (i) $(a')' = a$	6. Hukum penyerapan : (i) $a + (a \,.\, b) = a$ (ii) $a \,.\, (a + b) = a$
7. Hukum komulatif : (i) $a + b = b + a$ (ii) $a \,.\, b = b \,.\, a$	8. Hukum asosiatif : (i) $a + (b + c) = (a + b) + c$ (ii) $a \,.\, (b \,.\, c) = (a \,.\, b) \,.\, c$
9. Hukum distributif :	10. Hukum De Morgan :

(i) $a + (b \,.\, c) = (a + b) \,.\, (a + c)$ (ii) $a \,.\, (b + c) = (a \,.\, b) + (a \,.\, c)$	(i) $(a + b)' = a'b'$ (ii) $(a \,.\, b)' = a' + b'$
11. Hukum 0/1 : (i) $0' = 1$ (ii) $1' = 0$	

Diperoleh hukum – hukum aljabar Boolean dari hukum – hukum aljabar dengan cara mempertukarkan

$\cup$ dengan + atau $\vee$ dengan +

$\cap$ dengan . atau $\wedge$ dengan .

U dengan 1 atau **T** dengan 1

Ø dengan 0 atau **F** dengan 0

Perhatikanlah bahwa hukum yang ke-(ii) dari setiap hukum di atas merupakan dual dari hukum yang ke-(i). Sebagai contoh,

Hukum komutatif : $a + b = b + a$

dualnya : $a \,.\, b = b \,.\, a$

Hukum asosiatif : $a + (b + c) = (a + b) + c$

dualnya : $a \,.\, (b \,.\, c) = (a \,.\, b) \,.\, c$

Hukum distributif : $a + (b \,.\, c) = (a + b) \,.\, (a + c)$

dualnya : $a \,.\, (b + c) = (a \,.\, b) + (a \,.\, c)$

2.6 Fungsi Boolean

> (Definisi 2.4) Fungsi Boolean (disebut juga fungsi biner) adalah pemetaan dari B^n ke B melalui ekspresi Boolean, dapat dituliskan 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*. [3]

Misalkan ekspresi Boolean dengan *n* peubah adalah $E(x_1, x_2, ..., x_n)$. Menurut definisi 2.4, 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 Booleannya 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)

$= x'z \cdot 1 + xy'$ (Hukum komplemen)

$= x'z + xy'$ (Hukum identitas)

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

2.7 Komplemen Fungsi Boolean

Bila sebuah fungsi Boolean dikomplemenkan, makan akan memperoleh fungsi komplemen. Fungsi komplemen berguna ketika menyederhanakan fungsi Boolean. Fungsi komplemen dari suatu fungsi f, yaitu f' dapat dicari dengan dua cara berikut :

1. Cara pertama : menggunakan hukum De Morgan

   Hukum De Morgan untuk dua buah peubah, x_1 dan x_2 adalah

   (i) $(x_1 + x_2)' = x_1'x_2'$

   (ii) dan dualnya : $(x_1 \cdot x_2)' = x_1' + x_2'$

Hukum De Morgan untuk tiga buah peubah, x_1, x_2 dan x_3 adalah

(i) $(x_1 + x_2 + x_3)' = (x_1 + y')$, yang dalam hal ini $y = x_2 + x_3$

$= x_1'y'$

$= x1'(x_2 + x_3)'$

$= x_1'x_2'x_3'$

(ii) dan dualnya : $(x_1 . x_2 . x_3)' = x_1' + x_2' + x_3'$

Hukum De Morgan untuk *n* buah peubah, $x_1, x_2, ... ,x_n,$ adalah

(iii) $(x_1 + x_2 + ... + x_n)' = x_1' x_2' ... x_n'$

(iv) dan dualnya : $(x_1 . x_2 . ... . x_n)' = x_1' + x_2' + ... + x_n'$

2. Cara kedua : menggunakan prinsip dualitas.

Tentukan dual dari ekspresi Boolean yang merepresentasikan *f*, lalu komplemenkan setiap literal di dalam dual tersebut. Bentuk akhir yang diperoleh menyatakan fungsi komplemen. Sebagai contoh,

Misalkan $f(x, y, z) = x(y'z' + yz)$, maka dual dari ekspresi Booleannya adalah

$x + (y' + z') (y + z)$

Komplemenkan tiap literal dari dual di atas menjadi

$x' + (y + z) (y' + z') = f'$

Jadi, $f'(x, y, z) = x' + (y + z) (y' + z')$

2.8 Bentuk Kanonik

Ekspresi Boolean yang menspesifikasikan suatu fungsi dapat disajikan dalam dua bentuk. Pertama, sebagai penjumlahan dari hasil kali dan kedua sebagai perkalian dari hasil jumlah. Misalnya,

$f(x, y, z) = x'y'z + xy'z' + xyz$

dan

$g(x, y, z) = (x + y + z)\,(x + y' + z)\,(x + y' + z')\,(x' + y + z')\,(x' + y' + z)$

adalah dua buah fungsi yang sama (dapat ditunjukkan dari tabel kebenarannya). Fungsi yang pertama, *f*, muncul dalam bentuk penjumlahan dari hasil kali, sedangkan fungsi yang kedua, *g*, muncul dalam bentuk perkalian dari hasil jumlah. Perhatikan juga bahwa setiap suku (*term*) di dalam ekspresi mengandung literal yang lengkap dalam peubah *x*, *y* dan *z*, baik peubahnya tanpa komplemen maupun dengan komplemen. Ada dua macam bentuk *term*, yaitu *minterm* (hasil kali) dan *maxterm* (hasil jumlah).

Ekspresi Boolean yang dinyatakan sebagai penjumlahan dari satu atau lebih *minterm* atau perkalian dari satu atau lebih *maxterm* disebut dalam bentuk kanonik. Jadi, ada dua macam bentuk kanonik:

1. Penjumlahan dari hasil kali (*sum-of-product* atau SOP)
2. Perkalian dari hasil jumlah (*product-of-sum* atau POS)

Fungsi $f(x, y, z) = x'y'z + xy'z' + xyz$ dikatakan dalam bentuk SOP dan fungsi $g(x, y, z) = (x + y + z)\,(x + y' + z)\,(x + y' + z')\,(x' + y + z')\,(x' + y' + z)$ dikatakan dalam bentuk POS. Nama lain untuk SOP adalah bentuk normal

disjungtif (*disjunctive normal form*) dan nama lain POS adalah bentuk normal konjungtif (*conjunctive normal form*).

Minterm dilambangkan sebagai huruf *m* kecil berindeks. Indeks menyatakan nilai desimal dari *string* biner yang merepresentasikan *term*. Misalnya pada *term* dengan 2 peubah *x* dan *y*, indeks 0 pada m_0 menyatakan nilai desimal dari 00 ($x = 0$ dan $y = 0$), indeks 1 pada m_1 menyatakan nilai desimal dari 01 ($x = 0$ dan $y = 1$) dan seterusnya. Jadi, untuk *minterm* dari 3 peubah (*x*, *y*, dan *z*), jika ditulis m_6 maka ini berarti *minterm xyz*' karena 6 (desimal) = 110 (biner); di sini $x = 1$, $y = 1$ dan $z = 0$. Peubah *x* dan *y* dinyatakan tanpa komplemen sedangkan peubah *z* dinyatakan dengan komplemen karena bernilai 0, sehingga ditulis *xyz*'.

Maxterm dilambangkan sebagai huruf *M* besar berindeks. Indeks menyatakan nilai desimal dari *string* biner yang merepresentasikan $x + y$. Misalnya pada *term* dengan 2 peubah *x* dan *y*, indeks 0 pada M_0 menyatakan nilai desimal dari 00 ($x = 0$ dan $y = 0$), indeks 1 pada M_1 menyatakan nilai desimal dari 01 ($x = 0$ dan $y = 1$) dan seterusnya. Jadi, untuk *maxterm* dari 3 peubah (*x*, *y*, dan *z*), jika ditulis M_6 maka ini berarti *maxterm* $x' + y' + z$ karena 6 (desimal) = 110 (biner); di sini $x = 1$, $y = 1$ dan $z = 0$. Peubah *x* dan *y* dinyatakan dengan komplemen sedangkan peubah *z* dinyatakan tanpa komplemen karena bernilai 0, sehingga ditulis $x' + y' + z$.

Tabel 2.7 Tabel minterm dan maxterm dengan 2 peubah

x	*y*	*Minterm*		*Maxterm*	
		Suku	Lambang	Suku	Lambang
0	0	*x*'*y*'	m_0	*x* + *y*	M_0
0	1	*x*'*y*	m_1	*x* + *y*'	M_1
1	0	*xy*'	m_2	*x*' + *y*	M_2
1	1	*xy*	m_3	*x*' + *y*'	M_3

Tabel 2.8 Tabel minterm dan maxterm dengan 3 peubah

x	*y*	*z*	*Minterm*		*Maxterm*	
			Suku	Lambang	Suku	Lambang
0	0	0	*x*'*y*'*z*'	m_0	*x* + *y* + *z*	M_0
0	0	1	*x*'*y*'*z*	m_1	*x* + *y* + *z*'	M_1
0	1	0	*x*'*yz*'	m_2	*x* + *y*' + *z*	M_2
0	1	1	*x*'*yz*	m_3	*x* + *y*' + *z*'	M_3
1	0	0	*xy*'*z*'	m_4	*x*' + *y* + *z*	M_4
1	0	1	*xy*'*z*	m_5	*x*' + *y* + *z*'	M_5
1	1	0	*xyz*'	m_6	*x*' + *y*' + *z*	M_6
1	1	1	*xyz*	m_7	*x*' + *y*' + *z*'	M_7

Untuk membentuk fungsi dalam bentuk SOP, tinjau kombinasi nilai – nilai peubah yang memberikan nilai fungsi sama dengan 1. Misalkan kombinasi nilai – nilai peubah yang memberikan nilai fungsi sama dengan 1 adalah 001, 100, dan 111, maka bentuk SOP fungsi tersebut adalah:

$f(x, y, z) = x'y'z + xy'z' + xyz$

atau (dengan menggunakan lambang *minterm*) dapat ditulis

$f(x, y, z) = m_1 + m_4 + m_7 = \sum (1, 4, 7)$

Untuk membentuk fungsi dalam bentuk POS, tinjau kombinasi nilai – nilai peubah yang memberikan nilai fungsi sama dengan 0. Misalkan kombinasi nilai – nilai peubah yang memberikan nilai fungsi sama dengan 0 adalah 000, 010, 101, dan 110, maka bentuk POS fungsi tersebut adalah

$$f(x, y, z) = (x + y + z)\,(x + y' + z)\,(x' + y + z')\,(x' + y' + z)$$

atau (dengan menggunakan lambang *maxterm*) dapat ditulis

$$f(x, y, z) = M_0 + M_2 + M_5 + M_6 = \prod (0, 2, 5, 6)$$

Notasi $\sum$ dan $\prod$ berguna untuk mempersingkat penulisan ekspresi dalam bentuk SOP dan POS.

2.9 Konversi Antar Bentuk Kanonik

Fungsi Boolean dalam bentuk kanonik SOP dapat ditransformasi ke bentuk kanonik POS, demikian pula sebaliknya. Misalkan *f* adalah fungsi Boolean dalam bentuk SOP dengan tiga peubah :

$$f(x, y, z) = \sum (1, 4, 5, 6, 7)$$

dan f' adalah fungsi komplemen dari *f*,

$$f'(x, y, z) = \prod (0, 2, 3) = m_0 + m_2 + m_3$$

Dengan menggunakan hukum De Morgan, kita dapat memperoleh fungsi *f* dalam bentuk POS :

$$f'(x, y, z) = (f'(x, y, z))' = (m_0 + m_2 + m_3)'$$

$$= m_0' \,.\, m_2' \,.\, m_3'$$

$$= (x'y'z')' \,.\, (x'yz')' \,.\, (x'yz)'$$

$$= (x + y + z) \,.\, (x + y' + z) \,.\, (x + y' + z')$$

$= M_0\ M_2\ M_3$

$= \prod (0, 2, 3)$

Jadi, $f(x, y, z) = \sum (1, 4, 5, 6, 7) = \prod (0, 2, 3)$

2.10 Bentuk Baku

Dua bentuk kanonik adalah bentuk dasar yang diperoleh dengan membaca fungsi dari tabel kebenaran. Bentuk ini umumnya sangat jarang muncul, karena setiap suku (*term*) di dalam bentuk kanonik harus mengandung literal lengkap, baik dalam bentuk normal (x) atau dalam bentuk komplemennya (x').

Cara lain untuk mengekspresikan fungsi Boolean adalah bentuk baku (*standard*). Pada bentuk ini, suku – suku yang membentuk fungsi dapat mengandung satu, dua, atau sejumlah literal. Dua tipe bentuk baku adalah bentuk baku SOP dan bentuk baku POS. Contohnya,

$f(x, y, z) = y' + xy + x'yz$ (bentuk baku SOP)

$f(x, y, z) = x(y' + z)(x' + y + z')$ (bentuk baku POS)

Perbedaan antara bentuk kanonik dan bentuk baku adalah, pada bentuk kanonik, setiap *term* harus mengandung literal lengkap, sedangkan pada bentuk baku setiap *term* tidak mengandung literal lengkap.

Konversi ke bentuk baku:

f(x,y,z) = x + y’z

= x . 1 . 1 + 1 . y’z

= x (y + y’) (z + z’) + (x + x’) y’z

= x (yz + yz’ + y’z + y’z’) + xy’z + x’y’z

= xyz + xyz’ + xy’z + xy’z’ + xy’z + x’y’z

= m7 + m6 + m5 + m4 + m5 + m1

= ∑(1,4,5,6,7)

2.11 Penyederhanaan Fungsi Boolean

Fungsi Boolean seringkali mengandung operasi – operasi yang tidak perlu, literal atau suku – suku yang berlebihan. Oleh karena itu, kita dapat menyederhanakan fungsi Boolean lebih lanjut. Menyederhanakan fungsi Boolean artinya mencari bentuk fungsi lain yang ekivalen tetapi dengan jumlah literal atau operasi yang lebih sedikit. Penyederhanaan fungsi Boolean disebut juga minimisasi fungsi.

Dipandang dari segi aplikasi aljabar Boolean, fungsi Boolean yang lebih sederhana berarti rangkaian logikanya juga lebih sederhana (menggunakan jumlah gerbang logika lebih sedikit). Ada tiga metode yang dapat digunakan untuk menyederhanakan fungsi Boolean :

1. Secara aljabar, menggunakan hukum – hukum aljabar Boolean.
2. Metode Peta Karnaugh.
3. Metode Quine-McCluskey (metode tabulasi)

2.11.1 Penyederhanaan Fungsi Boolean Secara Aljabar

Jumlah literal di dalam sebuah fungsi Boolean dapat diminimumkan dengan trik manipulasi aljabar. Sayangnya, tidak ada aturan khusus yang harus diikuti yang akan menjamin menuju ke jawaban akhir. Metode yang tersedia adalah prosedur yang *cut-and-try* yang memanfaatkan postulat, hukum – hukum dasar, dan metode manipulasi lain yang sudah dikenal. Sebagai contoh,

$f(x, y, z) = xz' + y'z + xyz'$

$= xz' \cdot 1 + y'z + xyz'$ (Hukum identitas)

$= xz' (1 + y) + y'z$ (Hukum distributif)

$= xz' \cdot 1 + y'z$ (Hukum dominansi)

$f(x, y, z) = xz' + y'z$ (Hukum identitas)

2.11.2 Metode Peta Karnaugh

Metode Peta Karnaugh (atau *K-map*) merupakan metode grafis untuk menyederhanakan fungsi Boolean. Metode ini ditemukan oleh Maurice Karnaugh pada tahun 1953. Peta Karnaugh adalah sebuah diagram / peta yang terbentuk dari kotak – kotak (berbentuk bujursangkar) yang bersisian. Tiap kotak merepresentasikan sebuah *minterm*. Tiap kotak dikatakan bertetangga jika *minterm – minterm* yang merepresentasikannya berbeda hanya 1 buah literal.

Peta Karnaugh dapat dibentuk dari fungsi Boolean yang dispesifikasikan dengan ekspresi Boolean maupun fungsi yang direpresentasikan dengan tabel kebenaran.

2.11.2.1 Peta Karnaugh Dengan Dua Peubah

Misalkan dua peubah di dalam fungsi Boolean adalah *x* dan *y*. Baris pada peta Karnaugh untuk peubah *x* dan kolom untuk peubah *y*. Baris pertama diidentifikasi nilai 0 (menyatakan *x*'), sedangkan baris kedua dengan 1 (menyatakan *x*). Kolom pertama diidentifikasi nilai 0 (menyatakan *y*'), sedangkan kolom kedua dengan 1 (menyatakan *y*). Setiap kotak merepresentasikan *minterm* dari kombinasi baris dan kolom yang bersesuaian. Berikut terdapat tiga cara yang lazim digunakan sejumlah literatur dalam menggambarkan peta Karnaugh untuk dua peubah.

m_0	m_1
m_2	m_3

Gambar 2.9 Penyajian 1 - Peta Karnaugh dengan 2 peubah

		y	
		0	1
x	0	*x*'*y*'	*x*'*y*
	1	*xy*'	*xy*

Gambar 2.10 Penyajian 2 - Peta Karnaugh dengan 2 peubah

	y'	*y*
x'	*x*'*y*'	*x*'*y*
x	*xy*'	*xy*

Gambar 2.11 Penyajian 3 - Peta Karnaugh dengan 2 peubah

Perhatikan bahwa dua kotak yang bertetangga pada peta Karnaugh hanya berbeda satu bit atau satu literal.

2.11.2.2 Peta Karnaugh Dengan Tiga Peubah

Untuk fungsi Boolean dengan tiga peubah (misalkan *x*, *y* dan *z*), jumlah kotak di dalam peta Karnaugh meningkat menjadi $2^3 = 8$. Baris pada peta Karnaugh untuk peubah *x* dan kolom untuk peubah *yz*. Baris pertama diidentifikasi nilai 0 (menyatakan *x*'), sedangkan baris kedua dengan 1 (menyatakan *x*). Kolom pertama diidentifikasi nilai 00 (menyatakan *x*'*y*'), kolom kedua diidentifikasi nilai 01 (menyatakan *xy*'), kolom ketiga diidentifikasi 11 (menyatakan *xy*). Perhatikanlah bahwa antara satu kolom dengan kolom berikutnya hanya berbeda satu bit. Setiap kotak merepresentasikan *minterm* dari kombinasi baris dan kolom yang bersesuaian.

		yz			
		00	01	11	10
x	0	*x*'*y*'*z*'	*x*'*y*'*z*	*x*'*yz*	*x*'*yz*'
	1	*xy*'*z*'	*xy*'*z*	*xyz*	*xyz*'

Gambar 2.12 Peta Karnaugh dengan 3 peubah

2.11.2.3 Peta Karnaugh Dengan Empat Peubah

Misalkan empat peubah di dalam fungsi Boolean adalah *w*, *x*, *y* dan *z*. Jumlah kotak di dalam peta *Karnaugh* meningkat menjadi $2^4 = 16$. Baris pada peta Karnaugh untuk peubah *wx* dan kolom untuk peubah *yz*. Baris pertama diidentifikasi nilai 00 (menyatakan *w*'*x*'), baris kedua dengan 01 (menyatakan *w*'*x*), baris ketiga dengan 11 (menyatakan *wx*) dan baris keempat dengan 10 (menyatakan *wx*'). Kolom pertama diidentifikasi nilai 00 (menyatakan *y*'*z*'), kolom kedua diidentifikasi nilai 01 (menyatakan *yz*'), kolom ketiga diidentifikasi nilai 11 (menyatakan yz), sedangkan kolom keempat diidentifikasi dengan nilai 00 (menyatakan *yz*'). Setiap kotak merepresentasikan *minterm* dari kombinasi baris dan kolom yang bersesuaian.

		yz			
		00	01	11	10
wx	00	*w*'*x*'*y*'*z*'	*w*'*x*'*y*'*z*	*w*'*x*'*yz*	*w*'*x*'*yz*'
	01	*w*'*xy*'*z*'	*w*'*xy*'*z*	*w*'*xyz*	*w*'*xyz*'
	11	*wxy*'*z*'	*wxy*'*z*	*wxyz*	*wxyz*'
	10	*wx*'*y*'*z*'	*wx*'*y*'*z*	*wx*'*yz*	*wx*'*yz*'

Gambar 2.13 Peta Karnaugh dengan 4 peubah

2.11.3 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 :

$f(a, b, c, d) = \sum(3, 11) = \sum(0011, 1011) = a'b'cd + ab'cd$ dan

$f(a, b, c, d) = \sum(7, 11) = \sum(0111, 1011) = a'b'cd + ab'cd.$

	a b c d
3	0 0 1 1
11	1 0 1 1
BENTUK PRIMA -> (3,11)	- 0 1 1

Contoh (a)

	a b c d
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 fungsi 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. Dapat dilihat bahwa bentuk fungsi *output* selalu sama dengan bentuk fungsi *input*, artinya *input* dalam bentuk SOP akan menghasilkan *output* dalam bentuk SOP, dan demikian pula untuk bentuk POS.

Agar lebih jelas, perhatikan contoh berikut. Bentuk *input* dalam bentuk SOP :

$f(w, x, y, z) = \sum(1, 4, 6, 7, 8, 9, 10, 11, 15)$

Langkah – langkah minimisasi yang dilakukan adalah sebagai berikut:

(Langkah 1 dan langkah 2) Konversikan nilai *minterm* ke bentuk biner dengan panjang sebesar n peubah (4 bit) dan kelompokkan tiap minterm berdasarkan jumlah bit ‘1’ yang dimilikinya.

term	w	x	y	z	
1	0	0	0	1	→ (Jumlah bit ‘1’ = 1 buah).
4	0	1	0	0	
8	1	0	0	0	
6	0	1	1	0	→ (Jumlah bit ‘1’ = 2 buah).
9	1	0	0	1	
10	1	0	1	0	
7	0	1	1	1	→ (Jumlah bit ‘1’ = 3 buah).
11	1	0	1	1	
15	1	1	1	1	→ (Jumlah bit ‘1’ = 4 buah).

(Langkah 3 sampai 5) Kombinasikan *term* atau bentuk prima yang memiliki perbedaan tepat satu bit pada posisi yang sama. Hasil kombinasi

merupakan bentuk prima baru. Lakukan hingga didapat bentuk prima sesederhana mungkin. *Term* atau bentuk prima yang dikombinasikan diberi tanda '√'.

term	w	x	y	z	
1	0	0	0	1	√
4	0	1	0	0	√
8	1	0	0	0	√
6	0	1	1	0	√
9	1	0	0	1	√
10	1	0	1	0	√
7	0	1	1	1	√
11	1	0	1	1	√
15	1	1	1	1	√

term	w	x	y	z	
1,9	-	0	0	1	
4,6	0	1	–	0	
8,9	1	0	0	-	√
8,10	1	0	–	0	√
6,7	0	1	1	-	
9,11	1	0	–	1	√
10,11	1	0	1	-	√
7,15	-	1	1	1	
11,15	1	-	1	1	

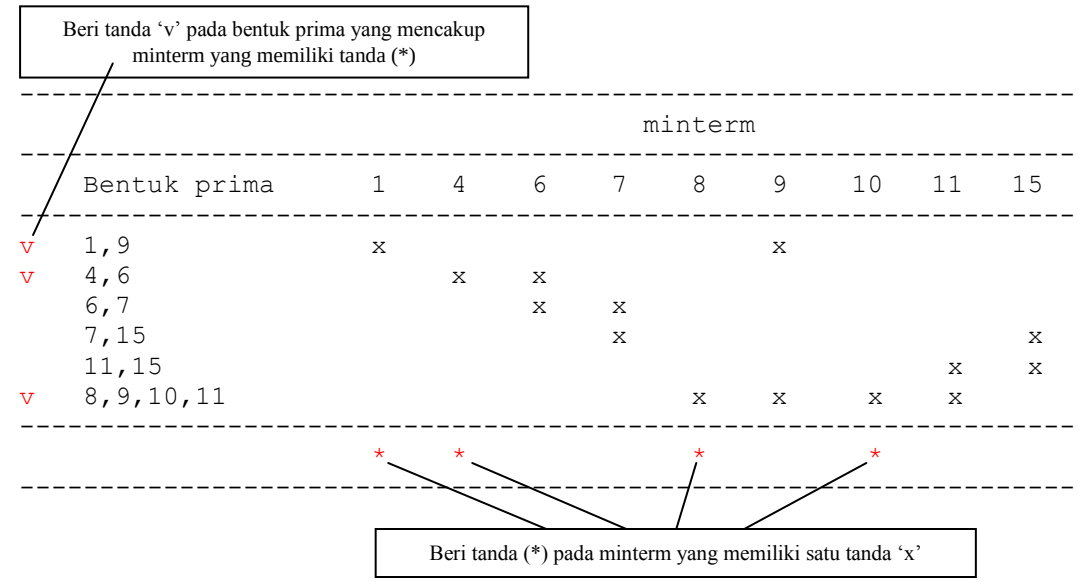
term	w	x	y	z	
8,9,10,11	1	0	-	-	√
8,10,9,11	1	0	-	-	√

(Langkah 6) Ambil semua bentuk prima yang tidak bertanda "√". Buat 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.

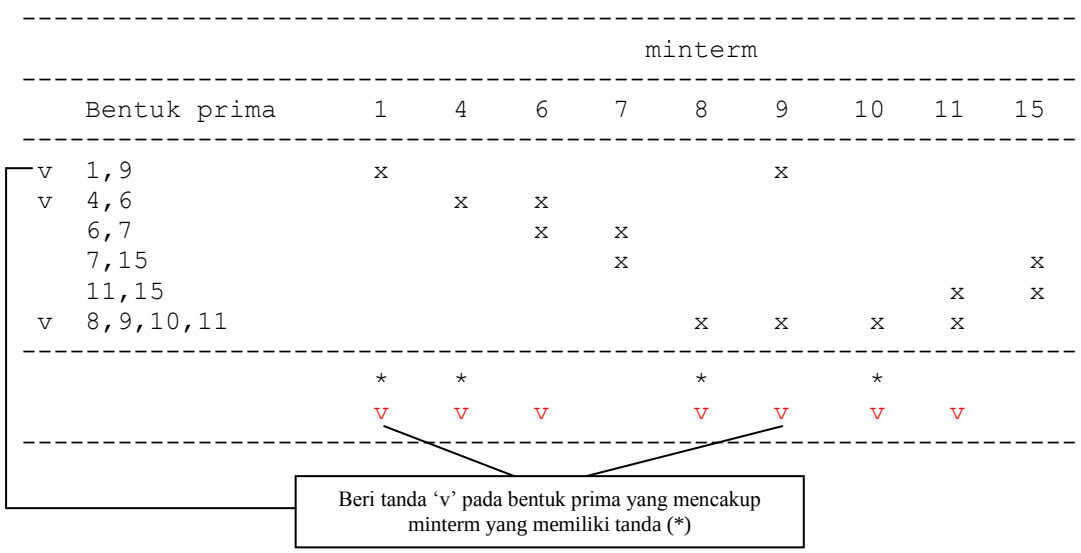
Minterm 1 dan 9 dicakup oleh bentuk prima (1,9) sehingga diberikan tanda 'x' pada minterm 1 dan 9

	minterm								
Bentuk prima	1	4	6	7	8	9	10	11	15
1,9	x					x			
4,6		x	x						
6,7			x	x					
7,15				x					x
11,15								x	x
8,9,10,11					x	x	x	x	

(Langkah 7.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.



(Langkah 7.b) Untuk setiap bentuk prima yang telah ditandai dengan “√”, beri tanda *minterm* yang dicakup oleh bentuk prima tersebut dengan tanda “√” (di baris bawah setelah “*”).



(Langkah 7.c dan 7.d) Sampai tahap ini, masih ada *minterm* yang belum tercakup dalam bentuk prima terpilih, yaitu 7, 15. Untuk mencakup *minterm* tersebut, maka dipilih bentuk prima (7,15), karena mencakup *minterm* 7 dan 15 sekaligus.

		minterm								
	Bentuk prima	1	4	6	7	8	9	10	11	15
v	1,9	x					x			
v	4,6		x	x						
	6,7			x	x					
v	7,15				x					x
	11,15								x	x
v	8,9,10,11					x	x	x	x	
		*	*			*		*		
		v	v	v	v	v	v	v	v	v

Semua minterm sudah tercakup dalam bentuk prima terpilih. Bentuk prima yang terpilih adalah :

1,9	yang bersesuaian dengan term $x'y'z$
4,6	yang bersesuaian dengan term $w'xz'$
7,15	yang bersesuaian dengan term xyz
8,9,10,11	yang bersesuaian dengan term wx'

Dengan demikian, fungsi Boolean hasil penyederhanaan dengan metode Quine-McCluskey adalah : $f(w, x, y, z) = x'y'z + w'xz' + xyz + wx'$.

2.12 Gerbang – Gerbang Sistem Digital Atau Gerbang Logika

Gerbang-gerbang sistem digital atau gerbang logika adalah piranti yang memiliki keadaan bertaraf logika. Gerbang logika dapat merepresentasikan keadaan dari bilangan biner.

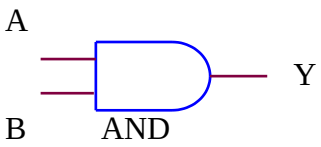
Terdapat dua keadaan pada gerbang logika, yaitu 0 dan 1. Tegangan yang digunakan dalam gerbang logika adalah HIGH (1) dan LOW (0). Sistem digital yang paling kompleks seperti komputer besar disusun dari gerbang logika dasar seperti AND, OR, NOT dan gerbang kombinasi (turunan) yang disusun dari gerbang dasar tersebut seperti NAND, NOR, EXOR, EXNOR.

Gerbang *Universal* merupakan salah satu gerbang turunan yang dirangkai sehingga menghasilkan output yang sama dengan output dari gerbang dasar maupun gerbang turunan. Adapun gerbang *universal* itu adalah NAND dan NOR.

Gerbang AND digunakan untuk menghasilkan logika 1 apabila semua masukan adalah berlogika 1. Gerbang OR digunakan untuk menghasilkan logika 1 apabila salah satu masukan berlogika 1. Gerbang not adalah gerbang pembalik (*inverter*). Output yang dihasilkan adalah kebalikan dari input yang diberikan. Gerbang logika turunan.

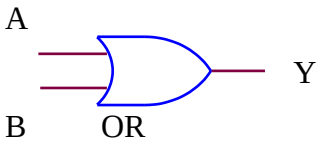
Gambar beberapa gerbang logika beserta table kebenarannya.[9]

Tabel 2.14 Tabel kebenaran gerbang AND



AND		
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

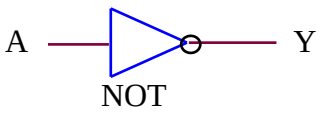
Tabel 2.15 Tabel kebenaran gerbang OR



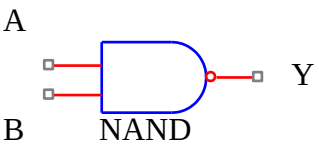
OR		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

T

Tabel 2.16 Tabel kebenaran gerbang NOT

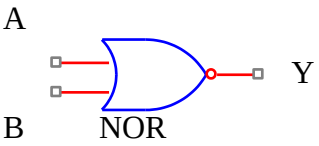


NOT	
A	Y
0	1
1	0



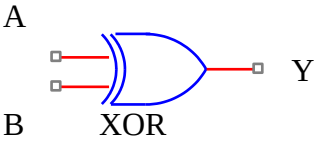
Tabel 2.17 Tabel kebenaran gerbang NAND

NAND		
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



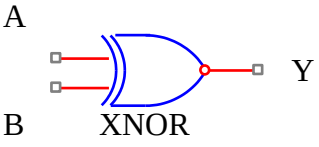
Tabel 2.18 Tabel kebenaran gerbang NOR

NOR		
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0



Tabel 2.19 Tabel kebenaran gerbang XOR

XOR		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0



Tabel 2.20 Tabel kebenaran gerbang XNOR

XNOR		
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

2.13 Logic Friday sebagai *tools generating* Gerbang Logika

Logic Friday adalah perangkat lunak yang dipergunakan sebagai alat (*tools*) untuk menganalisis ekspresi Boolean. Perangkat lunak ini lumrah digunakan dikalangan mahasiswa, ilmuwan, atau siapa saja yang berada dalam ruang lingkup sistem digital berdasarkan standart IC (*Integrated Circuits*).

Versi terbaru dari perangkat lunak ini adalam versi 1.1.3 yang di rilis pada 25 Maret 2011. Beberapa kinerja atau fasilitas yang didapat dari perangkat lunak Logic Friday ini diantaranya adalah:

i. Meng-inputkan ekspresi logika sebagai fungsi Boolean, table kebenaran, atau gerbang logika.

ii. Menghasilkan gerbang logika dari fungsi Boolean yang di inputkan.

iii. Membuat gerbang logika secara manual dengan *library* yang ada.

iv. Mengekspor dan mengimpor table kebenaran.

v. Menampilkan suatu fungsi sebagai table kebenaran, persamaan atau diagram gerbang logika.

Logic Friday menggunakan dua bahasa pemrograman yaitu *expresso* dan *misII*. Kedua bahasa pemrograman tersebut merupakan hak cipta dari *Regents of the University of California* tahun 1988-1993.

System Requiment untuk perangkat lunak Logic Fridat ini diantaranya:

i. Microsoft Windows 98, 2000, XP, Vista, Windows 7 (32 or 64 bit).

ii. *Pentium-class* PC, 450 MHz, 128 MB RAM, 10 MB *hard disk space.*

iii. Microsoft Internet Explorer versi 5 or setelahnya.