

# Ekuivalensi Aljabar *Boolean* dan *Decision Tree* dalam Perancangan Sistem Pengambil Keputusan

Ryan Samuel Chandra - 13521140<sup>1</sup>

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia

<sup>1</sup>13521140@std.stei.itb.ac.id

**Abstrak**—Di era modern, pengambilan keputusan didukung oleh teknologi komputer. Komputer dapat menarik pilihan terbaik berdasarkan data. Makalah ini membahas dua struktur yang cocok untuk mengambil keputusan, aljabar *boolean* dan pohon keputusan, dibuat dalam bentuk program serta rangkaian logika. Kedua struktur punya kelebihan dan kekurangan masing-masing. Maka dari itu, pemilihan struktur yang tepat akan mempermudah kita untuk merancang sebuah sistem pengambil keputusan.

**Kata Kunci**—Aljabar Boolean, Pohon Keputusan, Gerbang dan Rangkaian Logika, Bahasa Pemrograman C, Matematika Diskrit

**Abstract**—In modern eras, decision-making has been supported by computer technology. A computer can draw the best choice based on existing data. This paper discusses two structures suitable for making decisions, boolean algebra and decision trees, created in the form of programs and logic circuits. Both structures have their advantages and disadvantages. Therefore, choosing the correct structure will make it easier for us to design a decision-making system.

**Keywords**—Boolean Algebra, Decision Tree, Logical Gate and Circuit, C Programming Language, Discrete Mathematics

## I. PENDAHULUAN

Manusia diciptakan oleh Yang Mahakuasa dengan dilengkapi akal budi. Inilah yang membuat manusia berbeda dengan ciptaan lainnya, karena manusia bisa berpikir untuk membuat keputusan, mencari solusi, dan mengembangkan peradaban. Meskipun kita tidak mungkin mengetahui keseluruhan isi dunia secara pasti, sejauh ini ilmu pengetahuan telah berkembang dengan sangat dahsyat. Berbagai teori telah diciptakan oleh para ahli dengan latar belakang yang berbeda-beda, yang secara intensif mempelajari alam semesta dari segala sisi. Teori-teori tersebut bervariasi, baik yang bisa dipahami dengan logika sederhana, hingga yang kompleks dan mustahil untuk diamati tanpa pengetahuan yang cukup. Hal itu berdampak pada munculnya banyak penelitian baru untuk membuktikan atau mengembangkan teori-teori yang telah dikemukakan.

Salah satu bidang ilmu yang sedang mengalami proses pengembangan secara pesat adalah ilmu komputer. Menurut KBBI, komputer adalah alat elektronik otomatis yang dapat menghitung atau mengolah data secara cermat menurut instruksi, dan memberikan hasil pengolahan, serta dapat menjalankan sistem multimedia. Di zaman modern seperti ini,

semua kegiatan di dunia tidak terlepas dari peranan teknologi digital, yang sejatinya berakar dari program komputer. Terdapat bermacam-macam algoritma unik yang bertujuan membantu manusia dalam menyelesaikan tugas dengan cepat dan efisien.

Bagaimanapun juga, masalah pengambilan keputusan sering sekali terjadi dalam kehidupan sehari-hari. Saking pentingnya bagi manusia, kemampuan untuk memilih satu dari beberapa pilihan, terus diuji coba pada komputer. Bahkan, dibuat *artificial intelligence* pada penelitian tingkat lanjut. Meskipun sistem ini masih sangat terbatas, setidaknya komputer sudah bisa memilih secara akurat berdasarkan acuan pasti (berbeda dengan manusia yang mengacu pada akal budi dan hati nurani).



**Gambar 1** Ilustrasi Artificial Intelligence

(Sumber : <https://www.enterpriseai.news/2022/01/28/the-importance-of-humanized-autonomous-decision-making-in-ai/>)

Struktur data yang paling masuk akal bagi komputer untuk mengatasi permasalahan tersebut adalah pohon (*tree*). Dengan adanya percabangan yang terhingga, komputer bisa menentukan alur menuju opsi terbaik yang sesuai dengan kebutuhan, berdasarkan rentetan masukan *yes* atau *no* dari pengguna. Struktur ini mirip dengan susunan aljabar *boolean* yang merepresentasikan sebuah persamaan hanya dengan dua buah elemen, yaitu 0 dan 1. Kalkulasi dari sejumlah masukan akan memberikan keluaran sesuai dengan setelan yang sudah diatur sebelumnya.

Atas dasar keterkaitan tersebut, timbul pula rasa penasaran terhadap kesamaan kedua struktur, terutama perihal efisiensi mereka dalam mengambil keputusan. Baik *decision tree* maupun aljabar *boolean* dapat dikomputasikan sedemikian rupa. Maka dari itu, makalah ini dibuat agar dapat meninjau kedua struktur dari segi pemrograman, serta sebuah pemodelan khusus yang dapat mengotomatisasinya, yakni rangkaian logika.

## II. TINJAUAN PUSTAKA

### A. Aljabar Boolean

Pada tahun 1854, seorang matematikawan Inggris, George Boole, dalam bukunya yang berjudul "*The Laws of Thought*", memaparkan bahwa himpunan dan logika proposisi memiliki

sifat-sifat yang serupa. Aturan-aturan dasar ciptaannya yang kemudian dikenal sebagai logika *boolean*, membentuk stuktur matematika yang disebut aljabar *boolean* [1].

Pada tahun 1938, Claude Shannon memperlihatkan kegunaan aljabar *boolean* dalam merancang sirkuit yang menerima masukan 0 dan 1, dan menghasilkan keluaran yang serupa. Saat ini, aljabar *boolean* menjadi dasar teknologi komputer digital karena rangkaian elektronik dalam komputer (IC / *integrated circuit*) juga bekerja dengan metode operasi bit (0 dan 1). Selain itu, aljabar *boolean* digunakan secara luas dalam perancangan rangkaian pensaklaran serta rangkaian digital lainnya.

**DEFINISI.** 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, tuple:  $\langle B, +, \cdot, ', 0, 1 \rangle$  disebut aljabar *boolean*, jika untuk setiap  $a, b, c \in B$  berlaku aksioma berikut:

1. Identitas
  - (i)  $a + 0 = a$
  - (ii)  $a \cdot 1 = a$
2. Komutatif
  - (i)  $a + b = b + a$
  - (ii)  $a \cdot b = b \cdot a$
3. Distributif
  - (i)  $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
  - (ii)  $a + (b \cdot c) = (a + b) \cdot (a + c)$
4. Komplemen
 

Untuk setiap  $a \in B$ , terdapat elemen unik  $a' \in B$ ,

  - (i)  $a + a' = 1$
  - (ii)  $a \cdot a' = 0$

Aljabar himpunan dan logika proposisi juga merupakan aljabar *boolean* karena memenuhi keempat aksioma di atas. Kedua elemen unik dapat berbeda-beda, misalnya “F” (*false*) dan “T” (*true*) pada aljabar logika proposisi, atau “ $\emptyset$ ” (himpunan kosong) dan “U” (himpunan universal) pada aljabar himpunan. Secara umum, tetap digunakan “0” (disebut elemen *zero* atau minimum) dan “1” (disebut elemen *unit* atau maksimum).

**Tabel 1** Hukum-Hukum Aljabar Boolean [2]

1. Hukum Identitas (i) $a + 0 = a$ (ii) $a \cdot 1 = a$	2. Hukum Idempoten (i) $a + a = a$ (ii) $a \cdot a = a$
3. Hukum Komplemen (i) $a + a' = 1$ (ii) $a \cdot a' = 0$	4. Hukum Dominasi (i) $a \cdot 0 = 0$ (ii) $a + 1 = 1$
5. Hukum Involusi (i) $(a')' = a$	6. Hukum penyerapan (i) $a + ab = a$ (ii) $a(a + b) = a$
7. Hukum Komutatif (i) $a + b = b + a$ (ii) $a \cdot b = b \cdot a$	8. Hukum Asosiatif (i) $a + (b + c) = (a + b) + c$ (ii) $a(b \cdot c) = (a \cdot b) \cdot c$
9. Hukum Distributif (i) $a + (bc) = (a + b)(a + c)$ (ii) $a(b + c) = a \cdot b + a \cdot c$	10. Hukum De Morgan (i) $(a + b)' = a' \cdot b'$ (ii) $(ab)' = a' + b'$ (1)
11. Hukum 0/1 (i) $0' = 1$ (ii) $1' = 0$	

(1) Penulisan  $ab$  berarti  $a \cdot b$ , sedangkan  $a'$  (komplemen) bisa juga ditulis  $\bar{a}$ .

Fungsi *boolean* (disebut juga fungsi biner) merupakan pemetaan dari  $B^n$  ke  $B$  melalui ekspresi *boolean*. Dalam hal ini,  $B^n$  adalah himpunan yang beranggotakan pasangan terurut ganda- $n$  (*ordered n-tuple*) di dalam daerah asal  $B$  [1]. Sebagai contoh,  $f(x, y, z) = xyz + x'y + y'z$  adalah fungsi *boolean* yang memetakan nilai-nilai pasangan terurut ganda-3 ( $x, y, z$ ) ke himpunan  $\{0, 1\}$ . Jika ketiga nilai tersebut diubah menjadi 1, 0, dan 1, maka  $f(1, 0, 1) = 1 \cdot 0 \cdot 1 + 1' \cdot 0 + 0' \cdot 1 = 1$ .

**B. Peta Karnaugh dan Tabel Kebenaran**

Jumlah literal (seperti  $x, y, z$ , dan bentuk komplemennya) dalam fungsi *boolean* dapat disederhanakan dengan dua trik, Pertama, dengan metode manipulasi aljabar *boolean*, yaitu memanfaatkan hukum-hukum dasar. Kedua, dengan metode peta Karnaugh (*K-map*), yang ditemukan oleh Maurice Karnaugh pada tahun 1953 [1].

Peta Karnaugh merupakan sebuah diagram yang terbentuk dari kotak-kotak bersisian. Setiap kotak merepresentasikan sebuah *minterm*, yaitu suku hasil kali. Dalam *minterm*, bentuk tanpa komplemen menyatakan 1, sedangkan bentuk komplemen menyatakan 0. Misalkan nilai  $x'y'$  di kotak pertama,  $x'y$  di kotak kedua,  $xy'$  di kotak ketiga, dan  $xy$  di kotak keempat. Maka di kotak pertama berisi keluaran sebuah fungsi *boolean* ketika nilai  $x = 0$  dan  $y = 0$ , keluaran ketika  $x = 0$  dan  $y = 1$  di kotak kedua, demikian seterusnya. Sebagai catatan, dua kotak yang bertetangga harus memiliki perbedaan tepat satu buah literal.

*K-map* efektif digunakan hanya sampai 6 peubah (literal) saja. Jika peubah lebih dari 6, tidak lagi direkomendasikan untuk menggunakannya, karena komputasinya sangat tinggi [3]. Fungsi *boolean* dapat dituliskan dalam bentuk tabel kebenaran yang memuat setiap kombinasi peubah. Contoh:

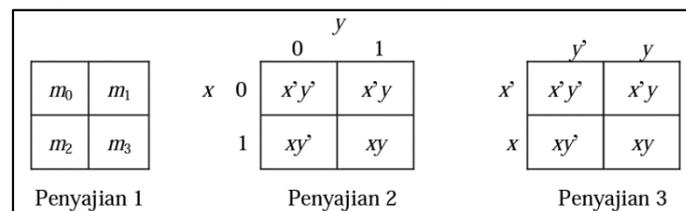
**Tabel 2** Contoh Tabel Kebenaran Dua Peubah

A	B	Minterm		Hasil $f(A,B)$
		Suku	Lambang	
0	0	$x'y'$	$m_0$	1
0	1	$x'y$	$m_1$	1
1	0	$xy'$	$m_2$	1
1	1	$xy$	$m_3$	0

akan menghasilkan peta Karnaugh sebagai berikut:

A \ B	0	1
0	1	1
1	1	0

Terlihat jelas bahwa struktur peta Karnaugh dua peubah jika ditinjau dari urutan *minterm* adalah sebagai berikut:



**Gambar 2** Tiga Opsi Penyajian Peta Karnaugh Dua Peubah

(Sumber : [https://informatika.stei.iib.ac.id/~rinaldi.munir/Matdis/2020-2021/Aljabar-Boolean-\(2020\)-bagian2.pdf](https://informatika.stei.iib.ac.id/~rinaldi.munir/Matdis/2020-2021/Aljabar-Boolean-(2020)-bagian2.pdf))

Tabel kebenaran dan peta Karnaugh untuk tiga, empat, dan lima peubah tentu saja berbeda dengan milik dua peubah.

Pengurutan posisi *minterm* tidak sesederhana penjelasan sebelumnya. Berikut ini adalah beberapa contoh:

**Tabel 3** Contoh Tabel Kebenaran Tiga Peubah

x	y	z	Minterm		f(x, y, z)
			Suku	Lambang	
0	0	0	$x'y'z'$	$m_0$	0
0	0	1	$x'y'z$	$m_1$	1
0	1	0	$x'yz'$	$m_2$	0
0	1	1	$x'yz$	$m_3$	0
1	0	0	$xy'z'$	$m_4$	1
1	0	1	$xy'z$	$m_5$	1
1	1	0	$xyz'$	$m_6$	0
1	1	1	$xyz$	$m_7$	1

akan menghasilkan fungsi *boolean*  $f(x, y, z) = x'y'z + xy'z' + xy'z + xyz$ , dan peta Karnaugh sebagai berikut:

x \ yz	00	01	11	10
0	0	1	0	0
1	1	1	1	0

		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 3** Penyajian Peta Karnaugh Tiga Peubah

(Sumber : [https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Ajabar-Boolean-\(2020\)-bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Ajabar-Boolean-(2020)-bagian2.pdf))

		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	11	$wxy'z'$	$wxy'z$	$wxyz$	$wxyz'$
	10	$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$

**Gambar 4** Penyajian Peta Karnaugh Empat Peubah

(Sumber : [https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Ajabar-Boolean-\(2020\)-bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Ajabar-Boolean-(2020)-bagian2.pdf))

	000	001	011	010	110	111	101	100
00	$m_0$	$m_1$	$m_3$	$m_2$	$m_6$	$m_7$	$m_5$	$m_4$
01	$m_8$	$m_9$	$m_{11}$	$m_{10}$	$m_{14}$	$m_{15}$	$m_{13}$	$m_{12}$
11	$m_{24}$	$m_{25}$	$m_{27}$	$m_{26}$	$m_{30}$	$m_{31}$	$m_{29}$	$m_{28}$
10	$m_{16}$	$m_{17}$	$m_{19}$	$m_{18}$	$m_{22}$	$m_{23}$	$m_{21}$	$m_{20}$

↑  
Garis pencerminan

**Dua kotak dianggap bertetangga jika secara fisik berdekatan dan merupakan pencerminan terhadap garis ganda**

**Gambar 5** Penyajian Peta Karnaugh Lima Peubah

(Sumber : [https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Ajabar-Boolean-\(2020\)-bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Ajabar-Boolean-(2020)-bagian2.pdf))

Penyederhanaan fungsi *boolean* dengan peta Karnaugh dilakukan dengan menggabungkan kotak-kotak yang bernilai 1 dan saling bersisian (hati-hati, syarat berbeda untuk lima peubah). Terdapat konsep penggulungan, yaitu kotak-kotak

yang terdapat pada sisi berseberangan (sisi kiri dengan sisi kanan, sisi atas dengan sisi bawah) adalah bertetangga juga. Kelompok nilai 1 tersebut dapat membentuk pasangan (dua), kuad (empat), dan oktet (delapan) [1]. Berikut contoh peta Karnaugh 4 peubah untuk meninjau masing-masing kelompok:

wx \ yz	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	1	1	1	1
10	0	1	1	1

Pada kelompok pertama (warna jingga), semua *minterm* memiliki kesamaan, yakni  $w = 1$  dan  $x = 1$ . Pada kelompok kedua (warna biru),  $w = 1$  dan  $z = 1$ . Pada kelompok ketiga (warna hijau),  $w = 1$  dan  $y = 1$ . Pada kelompok terakhir (warna merah),  $x = 1$ ,  $y = 1$ , dan  $z = 1$ . Maka, fungsi *boolean* paling sederhana yang terbentuk adalah:

$$f(w, x, y, z) = wx + wz + wy + xyz.$$

Sedikit tambahan, jika kesamaan terletak pada literal bernilai 0, maka literal tersebut ditulis ke persamaan dalam bentuk komplementnya. Contoh lain yang melibatkan hal tersebut beserta konsep penggulungan adalah sebagai berikut:

		yz			
		00	01	11	10
x	0	1	0	0	1
	1	1	1	0	1

**Hasil penyederhanaan:  $f(x, y, z) = z' + xy'$**

**Gambar 6** Contoh Penyederhanaan Fungsi Boolean

(Sumber : [https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Ajabar-Boolean-\(2020\)-bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Ajabar-Boolean-(2020)-bagian2.pdf))

Terdapat keadaan “*don't care*”, yaitu kondisi nilai peubah yang tidak diperhitungkan oleh fungsinya. Artinya, nilai 1 atau 0 dari peubah *don't care* tidak berpengaruh pada hasil fungsi tersebut [4]. Contohnya, peraga digital angka desimal 0 sampai 9 perlu merepresentasikan 4 bit sebanyak 10 buah. Namun, jika menggunakan empat peubah, terdapat 16 kombinasi. Maka, kombinasi ke-11 sampai ke-16 tidak terpakai.

w	x	y	z	Desimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

DONT CARE

**Gambar 7** Ilustrasi Keadaan *Don't Care* untuk Representasi Desimal

(Sumber : [https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Ajabar-Boolean-\(2020\)-bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Ajabar-Boolean-(2020)-bagian2.pdf))

Satu-satu nilai “X” (*don't care*) dalam peta Karnaugh dapat disamakan dengan 0 maupun 1, sehingga bisa diikutsertakan

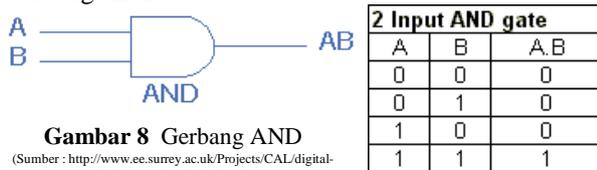
dalam pengelompokkan untuk penyederhanaan. Selain itu, pengelompokkan nilai satu sebaiknya dimulai dengan mencari oktet sebanyak-banyaknya terlebih dahulu, kemudian kuad, dan terakhir pasangan [4].

### C. Gerbang Logika

Komputer, atau perangkat elektronik lainnya, terdiri dari sejumlah sirkuit. Elemen dasar rangkaian disebut gerbang (*gate*), dan setiap jenisnya mewakili salah satu operasi *boolean*. Dengan gerbang serta aturan aljabar *boolean*, beragam rangkaian dapat disusun untuk memenuhi tugas tertentu.

Sirkuit sederhana memberikan *output* yang hanya bergantung pada input, bukan pada kondisi sirkuit saat ini. Dengan kata lain, sirkuit ini tidak memiliki kemampuan memori, dan disebut sirkuit kombinasional atau jaringan *gating* [8]. Berikut adalah 7 jenis gerbang logika sederhana:

#### 1. Gerbang AND

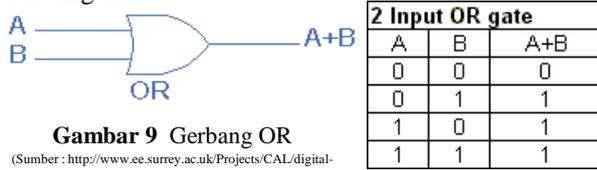


Gambar 8 Gerbang AND

(Sumber : <http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/index.html>)

Jika semua atau salah satu input merupakan 0, maka keluaran akan menjadi 0. Sedangkan jika semua input adalah 1, maka keluaran akan bernilai 1 [7].

#### 2. Gerbang OR

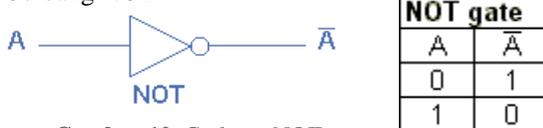


Gambar 9 Gerbang OR

(Sumber : <http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/index.html>)

Apabila semua atau salah satu input merupakan 1, maka *output* akan menjadi 1. Sedangkan jika semua input adalah 0, maka *output* adalah 0 [7].

#### 3. Gerbang NOT

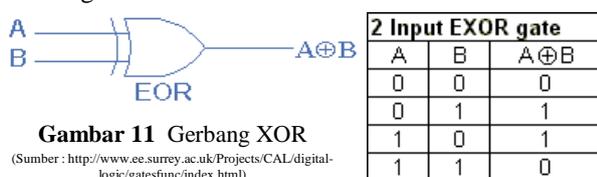


Gambar 10 Gerbang NOT

(Sumber : <http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/index.html>)

Fungsinya adalah sebagai *inverter* (pembalik). Nilai *output* akan berlawanan dengan inputnya [7].

#### 4. Gerbang XOR / EOR

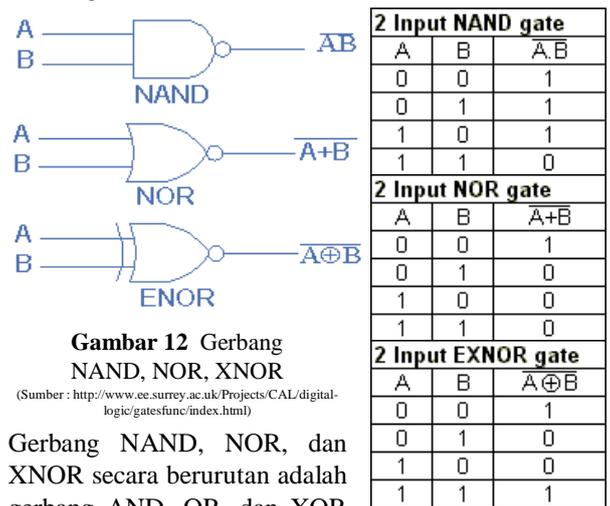


Gambar 11 Gerbang XOR

(Sumber : <http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/index.html>)

Apabila input berbeda (contoh: A=1, B=0) maka keluaran adalah 1. Sedangkan jika input adalah sama, maka keluaran adalah 0 [7].

#### 5. Gerbang NAND, NOR, dan XNOR/ENOR/EXNOR



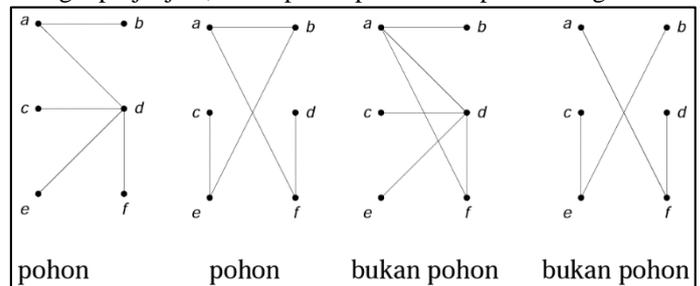
Gambar 12 Gerbang NAND, NOR, XNOR

(Sumber : <http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/index.html>)

Gerbang NAND, NOR, dan XNOR secara berurutan adalah gerbang AND, OR, dan XOR yang keluarannya dimasukkan lagi ke *inverter*, sehingga yang seharusnya 0 menjadi 1, dan demikian sebaliknya.

### D. Pohon dan Terminologinya

Pohon merupakan graf tak-berarah terhubung yang tidak mengandung sirkuit [9]. Pohon yang satu buah simpulnya diperlakukan sebagai akar dan sisi-sisinya diberi arah sehingga menjadi graf berarah dinamakan pohon berakar (*rooted tree*) [10]. Sebagai perjanjian, tanda panah pada sisi dapat diabaikan.

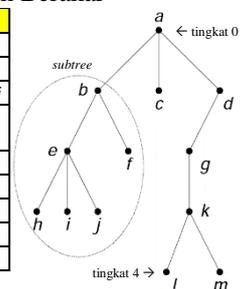


Gambar 13 Contoh Graf yang Berupa Pohon dan Bukan Pohon

(Sumber : [https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis2020-2021/Aljabar-Boolean-\(2020\)-bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis2020-2021/Aljabar-Boolean-(2020)-bagian2.pdf))

Tabel 4 Terminologi pada Pohon Berakar [10]

No	Istilah	Penjelasan
1	Anak ( <i>Child</i> )	<i>b, c, dan d</i> adalah anak dari <i>a</i>
2	Orang Tua ( <i>Parent</i> )	<i>a</i> adalah orang tua dari <i>b, c, d</i>
3	Lintasan ( <i>Path</i> )	Lintasan <i>a</i> ke <i>j</i> adalah <i>a, b, e, j</i>
4	Saudara Kandung ( <i>Sibling</i> )	<i>f</i> saudara kandung <i>e</i> , tapi <i>g</i> bukan, karena beda orang tua
5	Upapohon ( <i>Subtree</i> )	(perhatikan gambar)
6	Derajat ( <i>Degree</i> )	Jumlah anak sebuah simpul
7	Daun ( <i>Leaf</i> )	Simpul yang berderajat nol
8	Tingkat ( <i>Level</i> )	(perhatikan gambar)
9	Kedalaman ( <i>Depth</i> )	Tingkat maksimum pohon



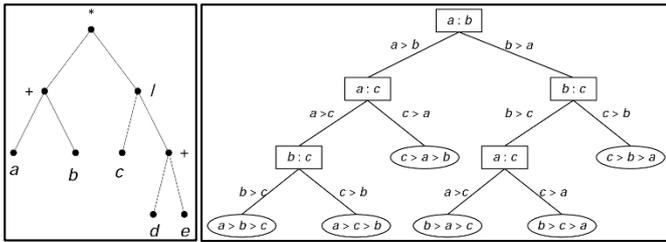
### E. Pohon Biner

Pohon *n*-ary adalah pohon berakar yang setiap simpulnya cabangnya memiliki paling banyak *n* buah anak. Pohon tersebut dikatakan teratur atau penuh (*full*) jika setiap simpul cabangnya mempunyai tepat *n* anak [10].

Pohon biner adalah pohon *n*-ary dengan *n* = 2. Maka, setiap simpulnya memiliki paling banyak 2 buah anak, dibedakan menjadi anak kiri (*left child*) dan anak kanan (*right child*). Pohon biner adalah pohon teratur, sebab urutan anak-anaknya penting (menjadi anak kiri dan anak kanan dianggap berbeda).

Contoh penerapan pohon biner adalah pada pohon ekspresi

dan pohon keputusan. Dalam pohon ekspresi, simpul daun diisi *operand*, sedangkan simpul dalam (simpul yang bukan simpul daun) diisi operator. Berikut adalah contohnya:



**Gambar 14** Pohon Ekspresi untuk  $(a+b)*(c/(d+e))$  dan Pohon Keputusan untuk Mengurutkan Tiga Buah Bilangan

(Sumber : [https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Ajabar-Boolean-\(2020\)-bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Ajabar-Boolean-(2020)-bagian2.pdf))

### F. Rekursivitas Pohon

Sebuah objek dikatakan rekursif jika ia didefinisikan dalam terminologi dirinya sendiri. Proses mendefinisikan objek itu disebut rekursi. Fungsi rekursif didefinisikan oleh dua bagian: basis (bagian yang berisi nilai fungsi yang terdefinisi secara eksplisit, bertindak sebagai penghenti fungsi rekursif), dan rekurens (mendefinisikan fungsi dalam terminologi dirinya sendiri, berisi kaidah untuk menemukan nilai fungsi pada suatu input dari nilai lainnya pada input yang lebih kecil) [11].

Pohon biner adalah struktur yang rekursif, sebab setiap simpul mempunyai cabang (upapohon) yang juga berupa pohon (biner) [11]. Oleh karena itu, pohon biner dapat didefinisikan secara rekursif sebagai berikut:

- Basis: pohon biner kosong adalah pohon biner.
- Rekurens: pohon biner tidak kosong terdiri dari sebuah simpul akar dan dua anak, upapohon kiri dan upapohon kanan, yang keduanya juga adalah pohon biner.

## III. IMPLEMENTASI DALAM BAHASA PEMROGRAMAN

### A. Fungsi dan Operasi Boolean

```
main.c | boolean.h | boolean.c
1 /* DEFINISI TIPE BOOLEAN */
2 #ifndef _BOOLEAN_H
3 #define _BOOLEAN_H
4 /* ----- */
5 #define boolean unsigned char
6 #define true 1
7 #define false 0
8
9 /* DEKLARASI FUNGSI BOOLEAN */
10 boolean AND (boolean A, boolean B);
11 /* Mengembalikan hasil operasi boolean A.B */
12 boolean OR (boolean A, boolean B);
13 /* Mengembalikan hasil operasi boolean A+B */
14 boolean NOT (boolean A);
15 /* Mengembalikan hasil boolean A' */
16 boolean XOR (boolean A, boolean B);
17 /* Mengembalikan hasil operasi boolean A XOR B */
18 boolean NAND (boolean A, boolean B);
19 /* Mengembalikan hasil operasi boolean (A.B)' */
20 boolean NOR (boolean A, boolean B);
21 /* Mengembalikan hasil operasi boolean (A+B)' */
22 boolean XNOR (boolean A, boolean B);
23 /* Mengembalikan hasil operasi boolean (A XOR B)' */
24
25 #endif
```

**Gambar 15** File “boolean.h” yang berisi header implementasi fungsi operasi boolean dalam file “boolean.c”

(Sumber : dokumentasi penulis)

Dalam bahasa pemrograman C, telah disediakan tiga buah operator dasar *boolean*, yaitu “&&” (·), “||” (+), dan “!” (·). Maka dari itu, fungsi AND, OR, dan NOT hanya digunakan untuk mengembalikan hasil operasi *boolean* A dan B dengan operator-operator tersebut. Fungsi lainnya diimplementasikan secara mandiri, memanfaatkan kombinasi 3 fungsi sebelumnya, sehingga sesuai dengan keluaran gerbang yang diinginkan.

### B. Struktur Data Pohon Biner

```
main.c | boolean.h | boolean.c | bintree.h | bintree.c | main2.c
1 #ifndef POHONBINER_H
2 #define POHONBINER_H
3
4 #include "boolean.h"
5
6 /* Selektor */
7 #define ROOT(p) (p)->info
8 #define LEFT(p) (p)->left
9 #define RIGHT(p) (p)->right
10
11 typedef char EType;
12 typedef struct treeNode* Address;
13 typedef struct treeNode {
14     EType info;
15     Address left;
16     Address right;
17 } TreeNode;
18
19 /* DEFINISI */
20 /* Pohon biner kosong: p = NULL */
21 /* Representasi Address dengan pointer */
22 /* EType adalah string */
23
24 typedef Address BinTree;
25
26 /* DEKLARASI FUNGSI DAN PROSEDUR */
27 BinTree NewTree (EType root, BinTree left_tree, BinTree right_tree);
28 /* Mengembalikan sebuah pohon biner dari root, left_tree, dan right_tree */
29
30 void CreateTree (EType root, BinTree left_tree, BinTree right_tree, BinTree *p);
31 /* I.S. Sembarang
32 F.S. Terbentuk sebuah pohon biner p dari root, left_tree, dan right_tree */
33
34 Address newTreeNode (EType val);
35 /* Mengembalikan Address p hasil lokasi sebuah simpul pohon
36 pt.info=val, pt.left=NULL, pt.right=NULL */
37
38 boolean isEmpty (BinTree p);
39 /* Mengirimkan true jika p adalah pohon biner yang kosong */
40
41 boolean isOneElmt (BinTree p);
42 /* Mengirimkan true jika p tidak kosong dan hanya terdiri atas 1 elemen */
43
44 void printTree (BinTree p, int h);
45 /* I.S. p terdefinisi, h adalah jarak indentasi (spasi) */
46 /* F.S. Semua simpul p sudah ditulis dengan indentasi (spasi) */
47
48 void keputusan (BinTree p);
49 /* Mengembalikan sebuah keputusan berdasarkan pohon p dan masukan pengguna */
50
51 #endif
```

**Gambar 16** File “bintree.h” yang berisi header implementasi struktur data pohon biner dalam file “bintree.c”

(Sumber : dokumentasi penulis)

*Tree* dibangun menggunakan *list* berkait. Menunjuk pada sebuah *address* (alamat memori) sama dengan menunjuk sebuah simpul. Setiap simpul memiliki tupel yang berisi nilai, penunjuk ke alamat anak kiri, dan penunjuk ke alamat anak kanan.

Semua *file* yang belum ditampilkan di sini bisa dilihat lengkapnya pada pranala di bagian **Lampiran**.

## IV. STUDI KASUS DAN ANALISIS

Misalkan ada sebuah gerai yang menjual satu jenis minuman. Setiap hari, gerai tersebut sangat ramai pengunjung, sampai para staf sering kewalahan dalam melayani. Meskipun demikian, bos dari gerai tersebut tetap sangat peduli terhadap kesehatan pelanggan, sehingga ia menerapkan beberapa aturan terkait pembelian minuman. Pertama, pembeli boleh memilih agar minumannya ditambahkan gula, jika ia suka manis, dan es, jika ia suka hidangan dingin. Kedua, pembeli yang menderita obesitas dilarang untuk memesan minuman bergula. Terakhir, pembeli yang sedang menderita flu dilarang untuk memesan minuman yang diberi es.

Agar pekerja-pekerjanya tidak kesulitan lagi, akhirnya bos



Bisa dilihat jelas fungsi GULA, ES, dan KELUAR pada “main.c” disesuaikan dengan tabel fungsi *boolean* (Gambar 17) yang terbentuk dari tabel kebenaran pada Gambar 20.

```
main.c | boolean.h | boolean.c | bintree.h | bintree.c | main2.c
1 #include <stdio.h>
2 #include "bintree.h"
3
4 int main() {
5     /* KAWAS */
6     BinTree beli_minuman, kiri, kanan;
7
8     /* ALGORITMA */
9     kiri = newTreeNode('M');
10    LEFT(kiri) = newTreeNode('0');
11    RIGHT(kiri) = newTreeNode('D');
12    LEFT(LEFT(kiri)) = newTreeNode('D');
13    RIGHT(LEFT(kiri)) = newTreeNode('D');
14    LEFT(RIGHT(kiri)) = newTreeNode('F');
15    RIGHT(RIGHT(kiri)) = newTreeNode('T');
16    LEFT(LEFT(LEFT(kiri))) = newTreeNode('F');
17    RIGHT(LEFT(LEFT(kiri))) = newTreeNode('T');
18    LEFT(RIGHT(LEFT(kiri))) = newTreeNode('F');
19    RIGHT(RIGHT(LEFT(kiri))) = newTreeNode('G');
20    LEFT(LEFT(RIGHT(kiri))) = newTreeNode('T');
21    RIGHT(LEFT(RIGHT(kiri))) = newTreeNode('E');
22    LEFT(LEFT(LEFT(LEFT(kiri)))) = newTreeNode('T');
23    RIGHT(LEFT(LEFT(LEFT(kiri)))) = newTreeNode('E');
24    LEFT(LEFT(RIGHT(LEFT(kiri)))) = newTreeNode('G');
25    RIGHT(LEFT(RIGHT(LEFT(kiri)))) = newTreeNode('C');
26
27    kanan = newTreeNode('K');
28
29    CreateTree('B', kiri, kanan, &beli_minuman);
30
31    keputusan(beli_minuman);
32 }
```

Gambar 19 Hasil Eksekusi Program Utama 2 (Sumber : dokumentasi penulis)

Pada program kedua, input “0” berarti *no* dan “1” berarti *yes*. Masukan diminta dari pengguna secara bertahap, mulai dari apakah ingin membeli, apakah suka manis, apakah obesitas, apakah suka hidangan dingin, dan apakah menderita flu. Menjawab “1” di pertanyaan suka manis adalah prasyarat masuk ke pertanyaan obesitas, dan menjawab “1” di pertanyaan suka hidangan dingin adalah prasyarat masuk ke pertanyaan flu.

Program kedua benar-benar dirancang agar mencerminkan pohon keputusan. Jawaban yang diberikan pada setiap simpul sangat menentukan pertanyaan selanjutnya yang muncul. Sedangkan, program pertama meminta semua jawaban serentak. Inilah perbedaan paling jelas dari kedua struktur pada saat mengambil keputusan, meskipun jawaban akhirnya sama saja.

Hal pertama yang bisa dibahas lebih lanjut adalah mengenai proses penyusunan. Penyusunan *tree* sampai sesuai dengan yang diinginkan lumayan sulit dan membutuhkan ketelitian, terutama

dalam bahasa pemrograman C. Alamat demi alamat dialokasi beserta isi elemennya yang hanya dapat berupa karakter (karena tidak ada *string* di C). Sebenarnya, bisa saja menggunakan banyak variabel seperti “kiri1”, “kanan1”, “kiri2”, “kiri22”, “kanan2”, “kanan22”, dan seterusnya, dengan angka sebagai nomor tingkat dan nomor orang tua. Barulah semua elemen disambung-sambungkan. Namun, hal tersebut tidak menjamin kepraktisan. Sebaliknya, penyusunan fungsi *boolean* sangatlah mudah, apalagi menggunakan aplikasi seperti Logic Friday.

Hal kedua yang perlu diperhatikan adalah kemangkusan. Terdapat dua jenis kemangkusan, yaitu dari sisi program (mesin / komputer) dan dari sisi pengguna. Secara teoritis, tipe *pointer* akan memakan memori sebesar 8 *byte* pada prosesor 64-bit, sedangkan *boolean (integer)* hanya 4 *byte*. Jelas penggunaan *tree* akan menghabiskan banyak sekali tempat pada saat membangun strukturnya. Bagaimana tidak, setiap simpul *tree* terdiri atas satu karakter (2 *byte*) atau *integer*, serta dua buah *pointer*! Di sisi lain, penggunaan tipe *boolean* hanya akan menempatkan sejumlah *integer* pada saat *runtime*.

Berbeda halnya dengan kemangkusan dari sisi pengguna. Jika menggunakan fungsi *boolean*, pengguna harus memasukkan tepat sejumlah angka untuk mendapatkan keluaran, misalnya 5 angka dalam kasus gerai minuman. Kelebihan atau kekurangan masukan akan berakibat program eror. Sedangkan pada *tree*, ada kemungkinan pengguna dimintai kurang dari 5 masukan (5 sebagai jumlah maksimalnya). Misalnya, saat seseorang tidak mau membeli, maka masukan sebuah “0” di awal akan langsung memberikan keluaran yang sesuai.

Jika diasumsikan satu fungsi adalah satu langkah, maka kompleksitas algoritma *tree* lebih baik, sebab kasus terburuknya adalah  $O(n)$ , yaitu memanggil fungsi “keputusan” secara rekursif sebanyak jumlah masukan ditambah 1 untuk akar. Pada struktur *boolean*, kasus terburuk adalah  $O(m \times n^2)$ , dengan  $m$  adalah jumlah keluaran (misalnya 3 dalam kasus gerai minuman, yaitu “gula”, “es”, dan “keluar”), dan  $n$  adalah jumlah masukan. Kuadrat pada jumlah masukan terjadi ketika semua masukan dioperasikan dalam masing-masing fungsi keluaran.

Hal ketiga yang ditinjau adalah kediskretan pertanyaan. Pada kasus gerai minuman, aspek yang ditinjau adalah diskret (tidak dependen). Maksudnya, ingin membeli minuman tidak ada hubungannya dengan suka manis dan obesitas, juga dengan suka hidangan dingin dan sedang flu. Maka dari itu, rangkaian logika menjadi sesuatu yang sangat efektif karena tidak mungkin ada kesalahan urutan input yang menyebabkan malafungsi.

Contoh kasus pertanyaan nondiskret adalah pemilihan jenis kamar hotel. Apabila seseorang memesan kamar untuk berdua, ia bisa memilih *single bed* atau *double bed*. Namun jika ia memilih kamar untuk menginap sendirian, ia otomatis mendapat *single bed*. Itulah alasan jumlah orang selalu menjadi pertanyaan pertama pada saat reservasi hotel. Maka dari itu, struktur *tree* lebih unggul kali ini, tetapi tetap tidak menutup kemungkinan solusi menggunakan aljabar *boolean*. Rangkaian logika untuk menangani kasus pertanyaan diskret dan nondiskret akan dibahas lebih detail pada bagian V.

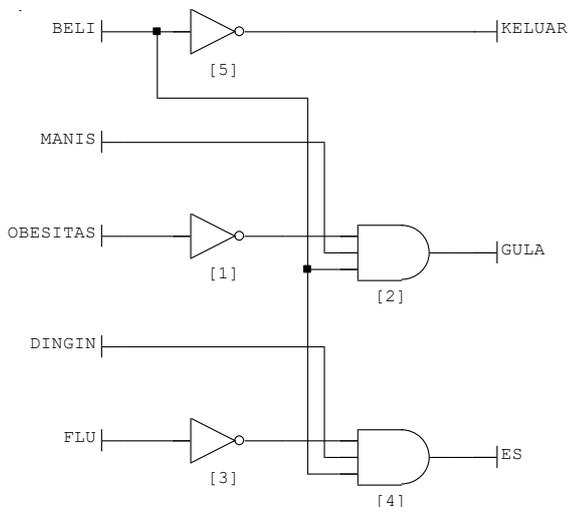
Hal terakhir adalah kediskretan pilihan. Dalam kasus ini, pembeli boleh memakai gula dan es secara bersamaan, sehingga komputasi pada *tree* akan jauh lebih mahal. Mengapa demikian? *Tree* mengharuskan semua simpul pada lintasan menuju simpul

daun dijawab secara kontinu. Ini menyebabkan pemrogram terpaksa mengalokasikan beberapa simpul daun yang isinya sama. Tinjau kembali **Gambar 17**, misalnya daun yang bertuliskan “TIDAK PAKAI GULA DAN ES” muncul sebanyak 4 kali, sebab ada 4 kemungkinan kasus yang memberikan jawaban itu. Komputer baru mengetahui penggunaan gula dan/atau es ketika semua pertanyaan selesai dijawab. Padahal, jika menggunakan *boolean*, dua fungsi saja (GULA dan ES) sudah cukup membuat gula dan es aktif. Lebih jelasnya bisa dilihat pada **Gambar 21**, gula dan es masing-masing hanya diberi satu lampu; bisa aktif bersamaan.

Tidak lupa dengan cerita, bos dari gerai minuman akhirnya paham dan puas dengan penelitian kecil yang ia lakukan. Selbihnya, penggunaan alat ia percayakan kepada para pekerja.

### V. PERANCANGAN RANGKAIAN LOGIKA

Term	BELI	MANIS	OBESITAS	DINGIN	FLU	=>	GULA	ES	KELUAR
0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	1	0	0	0	1
2	0	0	0	1	0	0	0	0	1
3	0	0	0	1	1	0	0	0	1
4	0	0	1	0	0	0	0	0	1
5	0	0	1	0	1	0	0	0	1
6	0	0	1	1	0	0	0	0	1
7	0	0	1	1	1	0	0	0	1
8	0	1	0	0	0	0	0	0	1
9	0	1	0	0	1	0	0	0	1
10	0	1	0	1	0	0	0	0	1
11	0	1	0	1	1	0	0	0	1
12	0	1	1	0	0	0	0	0	1
13	0	1	1	0	1	0	0	0	1
14	0	1	1	1	0	0	0	0	1
15	0	1	1	1	1	0	0	0	1
16	1	0	0	0	0	0	0	0	0
17	1	0	0	0	1	0	0	0	0
18	1	0	0	1	0	0	1	0	0
19	1	0	0	1	1	0	0	0	0
20	1	0	1	0	0	0	0	0	0
21	1	0	1	0	1	0	0	0	0
22	1	0	1	1	0	0	1	0	0
23	1	0	1	1	1	0	0	0	0
24	1	1	0	0	0	1	1	0	0
25	1	1	0	0	1	1	1	0	0
26	1	1	0	1	0	1	1	0	0
27	1	1	0	1	1	1	1	0	0
28	1	1	1	0	0	0	0	0	0
29	1	1	1	0	1	0	0	0	0
30	1	1	1	1	0	0	1	0	0
31	1	1	1	1	1	0	0	0	0

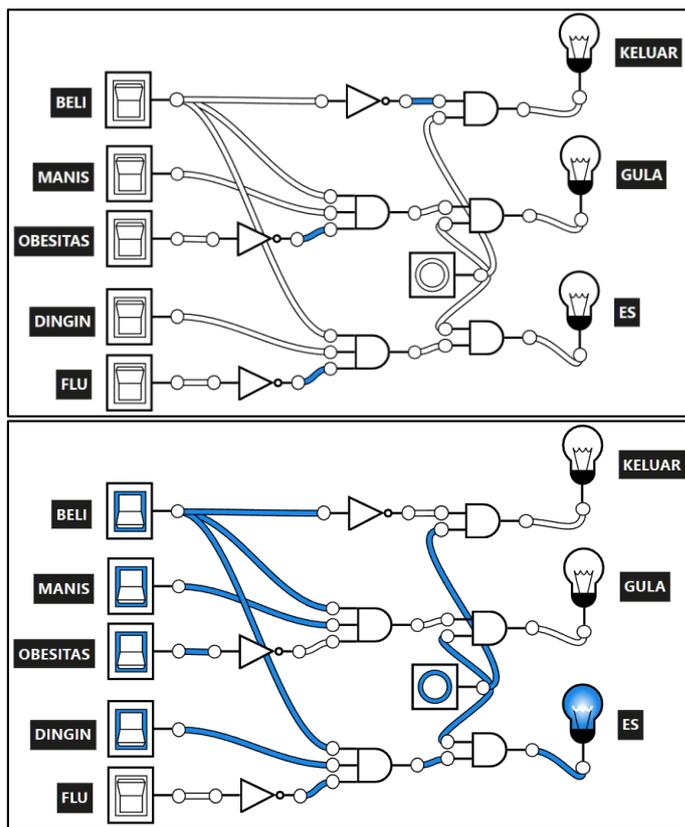


**Gambar 20** Tabel Kebenaran Asli dan Rangkaian-Logika Formal (Sumber : dokumentasi penulis)

Tabel dan rangkaian di atas disusun menggunakan aplikasi Logic Friday. Aplikasi tersebut dapat membuat tabel kebenaran dengan jumlah input yang diinginkan ( $\geq 2$ ). Setelah itu, tinggal

ditentukan keluaran untuk setiap fungsinya. Pada tabel di **Gambar 20**, ada 3 fungsi, yaitu GULA, ES, dan KELUAR, dengan masing-masing 5 parameter (*beli, manis, obesitas, dingin, flu*), seperti yang diterapkan pada bahasa C di bagian sebelumnya (**Gambar 18**). Selanjutnya, aplikasi merancang rangkaian logika sederhana yang sesuai dengan tabel tersebut. Ternyata, rangkaian untuk gerai minuman dapat disusun dengan hanya menggunakan dua buah gerbang AND 3-input dan tiga buah gerbang NOT.

Satu hal yang menjadi perhatian adalah perbedaan pengisian “0” dan *don't care* (“X”) pada keluaran fungsi. Untuk kasus ini, tidak ada *don't care*, sebab semua keluaran adalah wajib sebagaimana mestinya. Pada saat pembuatan, penulis sempat mengisi “X” di kolom GULA dan ES pada saat kolom BELI bernilai 0. Efeknya, keluaran fungsi GULA dan ES tetap dapat bernilai 1 meskipun input pada parameter *beli* bernilai 0. Hal tersebut tidak diinginkan, sebab tidak beli berarti keluar saja.



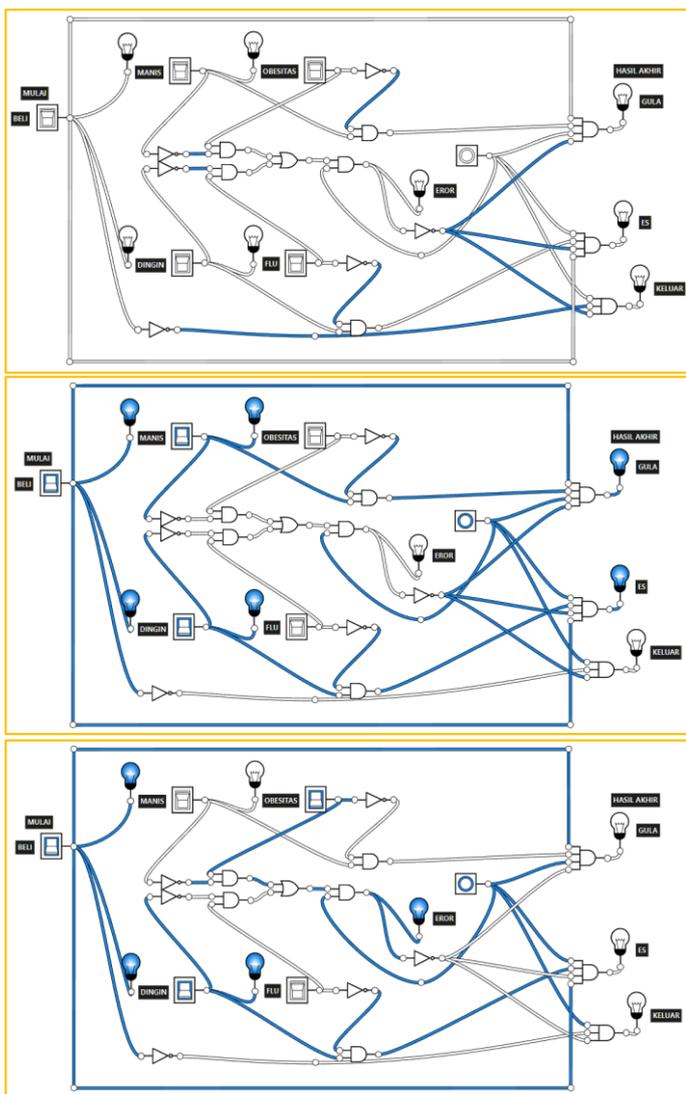
**Gambar 21** Rangkaian Logika Gerai Minuman (Versi 1) (Sumber : dokumentasi penulis)

Rangkaian pada gambar di atas disusun menggunakan aplikasi logic.ly berbasis web. Rangkaian tersebut meminta pengguna untuk menekan setiap saklar di bagian kiri, apabila ingin menyalakannya dan menyatakan “ya” atau “1” (sesuai dengan aljabar *boolean*). Menekan *push button* adalah langkah terakhir setelah mengatur semua saklar. Lampu *output* yang sesuai akan menyala bersamaan dengan itu. Pada praktiknya, menekan *push button* berarti mengaktifkan mesin otomatis pembuat minuman, jika lampu “KELUAR” tidak menyala.

Seperti yang sudah dibahas pada bagian **IV**, salah satu kelemahan dari rangkaian logika adalah ketidakmampuan untuk mengatur sebuah input yang hanya bisa dimasukkan setelah input lainnya. Dengan kata lain, semua input akan dibaca secara bersamaan tanpa memerhatikan koneksi.

Kita kembali pada persoalan gerai minuman. Andaikan orang yang obesitas haruslah orang yang suka manis, dan orang yang menderita flu haruslah orang yang suka hidangan dingin. Untuk itu, harus dilakukan sesuatu pada fungsi *boolean* dan rangkaian logika agar dapat mencegah kesalahan urutan input. Misalkan pengguna mengisi bagian flu dengan “1”, padahal ia tidak suka hidangan dingin, maka mesin minuman tidak akan berproses.

Idenya, dibuat satu fungsi lagi, yaitu “EROR”. Keluaran fungsi ini bernilai 1 apabila parameter *flu* bernilai 1 saat *dingin* bernilai 0, atau parameter *obesitas* bernilai 1 saat *manis* bernilai 0. Selanjutnya dihubungkan dengan fungsi lain, sehingga fungsi lain baru bisa bernilai 1 jika keluaran fungsi EROR bernilai 0. Demi kemudahan bagi pembeli, ditambahkan juga beberapa lampu selain *output*. Lampu yang menyala di sebelah sebuah saklar menandakan saklar tersebut harus diatur nilainya. Misalnya, lampu pada saklar “OBESITAS” akan menyala saat saklar “MANIS” dinyalakan.



**Gambar 22** Rangkaian Logika Gerai Minuman (Versi 2)  
(Sumber : dokumentasi penulis)

Urutan pengubahan rangkaian di atas adalah: BELI (terpisah); MANIS | DINGIN, OBESITAS | FLU. Seperti yang sudah disebutkan sebelumnya, lampu EROR akan menyala saat *push button* ditekan kapan pun sebuah parameter bernilai 1 sedangkan predesesornya dalam urutan pengisian bernilai 0. Jadi, jika kita mengajukan pertanyaan yang tidak diskret, rangkaian logika

menjadi lebih rumit. Dalam kasus ini, membutuhkan 6 buah gerbang NOT, 5 buah gerbang AND 2-input, 1 buah gerbang AND 3-input, 2 buah gerbang AND 3-input, sebuah gerbang OR 2-input, serta tambahan 5 buah lampu.

Persoalan gerai minuman yang begitu sederhana telah menggunakan tabel kebenaran 5-peubah, dengan 3 sampai 4 fungsi, menghasilkan rangkaian logika yang amat kompleks. Apa yang terjadi jika digunakan 6 peubah? Misalnya flu dibagi menjadi flu ringan dan flu berat. Secara teori kondisi tersebut (enam peubah) sudah mencapai batas maksimal keefektifitasan komputasi tabel kebenaran. Selain itu, rangkaian logika akan membutuhkan tambahan gerbang untuk mendeteksi kesalahan.

## VI. KESIMPULAN

Manusia adalah ciptaan yang spesial, karena dapat membuat keputusan berdasarkan hati nurani dan akal budi. Sebaliknya, komputer dapat membuat keputusan berdasarkan masukan, atau kebiasaan masukan dari sejumlah pengguna. Paling tidak, ada dua struktur data dasar yang dapat membantu komputer dalam melakukannya, yaitu aljabar *boolean* dan *decision tree* (pohon keputusan).

Keputusan yang dihasilkan dengan struktur *boolean* berupa *output* 0/1 untuk setiap objek pilihan. “0” berarti pilihan tersebut tidak diambil, sedangkan “1” sebaliknya. *Output* mengandalkan input yang juga berupa nilai 0 atau 1, untuk kemudian diproses dalam fungsi tertentu yang mengikuti hukum aljabar *boolean*. Pada *tree*, dihasilkan keluaran terang-terangan yang memuat pilihan terbaik saja. Struktur tersebut mengandalkan masukan “ya” dan “tidak” untuk setiap pertanyaan.

Ada setidaknya lima hal yang bisa dibandingkan, yaitu kesulitan proses penyusunan, kemangkusan dari sisi program dan pengguna, kompleksitas algoritma, kediskretan pertanyaan, serta kediskretan pilihan. Dalam mengambil keputusan sederhana, aljabar *boolean* bisa dikatakan unggul dibandingkan *tree* hanya pada aspek pertama, kedua, dan kelima.

Fungsi *boolean* dapat dirancang dalam sebuah pemodelan khusus yang disebut rangkaian logika. Bagaimanapun juga, perancangan akan semakin sulit dan membutuhkan banyak tambahan apabila jumlah fungsi atau peubah meningkat. Namun, *tree* mungkin malah membutuhkan lebih banyak perubahan pada percabangan ketika ada penambahan opsi beserta pertanyaan (peubah pada aljabar *boolean* sama dengan jumlah maksimum jenis pertanyaan pada *tree*).

Meskipun mirip, ekuivalensi aljabar *boolean* dan *decision tree* dalam mengambil keputusan masih lumayan rendah. Akan tetapi, keduanya mempunyai kelebihan dan kekurangan masing-masing, yang menjadikan mereka unik dan cocok untuk diterapkan sebagai pengambil keputusan.

## VII. LAMPIRAN

Kode program pengambil keputusan menggunakan struktur *boolean* maupun *decision tree* untuk kasus gerai minuman dapat diperiksa pada pranala berikut:

<https://github.com/RyanSC06/StrukturBoolean-dan-DecisionTree-untuk-MengambilKeputusan.git>

## VIII. UCAPAN TERIMA KASIH

Pertama-tama, penulis memanjatkan puji dan syukur kepada Tuhan Yang Maha Esa, atas hikmat dan kelancaran yang telah diberikan-Nya kepada penulis, sehingga makalah ini dapat selesai dengan baik dan tepat waktu. Penulis juga mengucapkan terima kasih kepada kedua orang tua, serta teman-teman yang sudah mendukung / memotivasi selama pembuatan makalah ini.

Kemudian, penulis juga berterima kasih secara khusus kepada Ibu Dr. Fariska Zakhralatifa Ruskanda, S.T., M.T., selaku dosen pengajar mata kuliah IF2120 Matematika Diskrit di kelas K2 pada tahun 2022, yang telah memberikan ilmu dan kesempatan kepada penulis untuk mengerjakan makalah ini.

## REFERENSI

- [1] Munir, Rinaldi. 2010. *Matematika Diskrit (Edisi 3)*. Bandung: Informatika Bandung.
- [2] Munir, Rinaldi. 2020. *Aljabar Boolean (Bagian 1)*. Diakses dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Aljabar-Boolean-\(2020\)-bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Aljabar-Boolean-(2020)-bagian1.pdf) tanggal 8 Desember 2022.
- [3] Jati. 12 Juni 2013. *Memahami Peta Karnaug (1)*. Diakses dari <https://jati.ida.ac.id/2013/06/memahami-peta-karnaugh-1.html> tanggal 8 Desember 2022.
- [4] Munir, Rinaldi. 2020. *Aljabar Boolean (Bagian 2)*. Diakses dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Aljabar-Boolean-\(2020\)-bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Aljabar-Boolean-(2020)-bagian2.pdf) tanggal 9 Desember 2022.
- [5] Munir, Rinaldi. 2020. *Aljabar Boolean (Bagian 3)*. Diakses dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Aljabar-Boolean-\(2020\)-bagian3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Aljabar-Boolean-(2020)-bagian3.pdf) tanggal 9 Desember 2022.
- [6] Sangosanya, Wale dkk. 1997-2005. *Basic Gates and Functions*. Diakses dari <http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/index.html> pada tanggal 9 Desember 2022.
- [7] Kosasi, Martin. 23 Januari 2017. *Pengertian Dan Jenis Gerbang Logika (Logic Gates)*. Diakses pada tanggal 9 Desember 2022 dari <https://www.logicgates.id/blogs/news/pengertian-dan-jenis-gerbang-logika-logic-gates>.
- [8] Rosen, Kenneth H. 2012. *Discrete Math and Its Applications (Seventh Edition)*. New York: McGraw-Hill.
- [9] Munir, Rinaldi. 2020. *Pohon (Bagian 1)*. Diakses dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag1.pdf> tanggal 9 Desember 2022.
- [10] Munir, Rinaldi. 2021. *Pohon (Bagian 2)*. Diakses dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2021-2022/Pohon-2021-Bag2.pdf> tanggal 9 Desember 2022.
- [11] Munir, Rinaldi. Tanpa tanggal. *Rekursi dan Relasi Rekurens (Bagian 1)*. Diakses dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Rekursi-dan-relasi-rekurens-\(Bagian-1\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Rekursi-dan-relasi-rekurens-(Bagian-1).pdf) pada 9 Desember 2022.

## PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Desember 2022



Ryan Samuel Chandra  
(13521140)