Aplikasi Fungsi Rekursif dan Pohon Biner dalam Algoritma Penyederhanaan Fungsi Boolean Menggunakan B-Trail

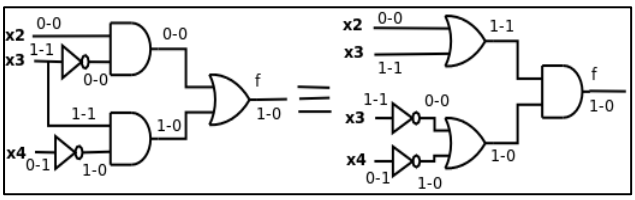
Ahmad Hasan Albana - 13522041[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522041@std.stei.itb.ac.id

***Abstract*—B-Trail adalah salah satu bentuk pemetaan fungsi boolean kedalam suatu angka biner dengan panjang 2^n, dengan n adalah banyaknya variabel Aljabar Boolean. Setiap *term* pada fungsi boolean memiliki nilainya masing-masing berdasarkan representasi binernya, dan nilai-nilai tersebut akan dipetakan ke bentuk B-Trail. Makalah ini membahas tentang aturan dan algoritma penyederhanaan fungsi boolean dalam bentuk kanonik *Sum of Product* dengan memanfaatkan bentuk B-Trail menggunakan rekursi dan pohon biner.**

***Keywords*—Angka Biner, B-Trail, Fungsi Boolean, Pohon Biner, Rekursi.**

I. PENDAHULUAN

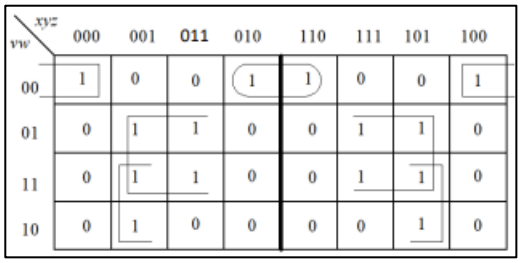
Fungsi Boolean adalah suatu fungsi yang dibentuk oleh *n* buah variabel Aljabar Boolean yang hanya dapat bernilai 1 (True) atau 0 (False). Secara umum, fungsi boolean dapat diekspresikan menggunakan 2 bentuk kanonik, yaitu *Sum of Product* (SOP) dan *Product of Sum* (POS). Bentuk SOP adalah fungsi boolean yang berisi penjumlahan dari beberapa perkalian semua variabel yang ada. Sedangkan, bentuk SOP adalah fungsi boolean yang berisi perkalian dari beberapa penjumlahan semua variabel yang ada.



***Gambar** 1: Gerbang Logika Fungsi Boolean Bentuk Kanonik POS (Kiri) dan SOP (Kanan)*

(Sumber: https://edwidianto.wordpress.com/2017/03/10/aljabar-boolean-dan-sintesis-rangkaian-logika-2/, diakses pada 7 Desember 2023)

Fungsi Boolean dapat disederhanakan dengan 3 cara, salah satunya yang terkenal adalah dengan Karnaugh Map (K-Map). Namun, cara tersebut akan sulit digunakan untuk jumlah variabel Boolean yang banyak dikarenakan harus memperhatikan posisi tabel dan pola secara manual.



Gambar 2: Peta Karnaugh dengan 5 variabel

(Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/12-Aljabar-Boolean-(2023)-bagian2.pdf, diakses pada 10 Desember 2023)

Angka biner pada Fungsi Boolean sangat erat kaitannya dengan dunia komputasi yang dapat menyelesaikan berbagai algoritma secara cepat. Oleh karena itu, dikembangkanlah metode penyederhanaan menggunakan angka biner, yaitu B-Trail dengan tujuan dapat menyelesaikan penyederhanaan aljabar boolean dengan jumlah variabel yang besar menggunakan bantuan program komputer. B-Trail adalah bentuk penulisan fungsi boolean menggunakan suatu angka biner yang memiliki panjang 2^n dengan n adalah banyaknya variabel Aljabar Boolean.

Penyederhanaan fungsi boolean dengan menggunakan B-Trail ini adalah metode yang dikembangkan dengan memanfaatkan penamaan *term* pada SOP yang menggunakan representasi angka binernya dan sifat angka biner seperti memiliki karakteristik yang sama untuk panjang angka berapapun sehingga dapat diterapkannya rekursif dan pohon biner beserta sifat-sifat angka biner lainnya.

Makalah ini membahas tentang aturan dan algoritma dari penyederhanaan Fungsi Boolean dengan B-Trail. Fungsi Boolean yang digunakan berfokus pada bentuk kanonik SOP untuk memperkecil ruang lingkup dengan asumsi konversi antar bentuk kanonik dapat dilakukan secara mandiri.

II. LANDASAN TEORI

A. Aljabar Boolean

Aljabar Boolean adalah suatu aturan dasar logika yang dibentuk menjadi suatu struktur matematika berupa tuple yang terdiri dari suatu himpunan variabel, 2 operator biner, sebuah operator uner, dan 2 elemen yang berbeda dari variabel. Aljabar

Boolean ini ditemukan oleh George Boole pada tahun 1854. Dengan B adalah himpunan variabel, tuple < B, +, ., ‘, 0, 1 > adalah salah satu contoh dari Aljabar Boolean karena memenuhi aksioma berikut dengan $\alpha, \beta, \gamma \in B$.

1. Identitas

$(i)\ \alpha + 0 = \alpha$

$(ii)\ \alpha . 1 = \alpha$

2. Komutatif

$(i)\ \alpha + \beta = \beta + \alpha$

$(ii)\ \alpha . \beta = \beta . \alpha$

3. Distributif

$(i)\ \alpha . (\beta + \gamma) = (\alpha . \beta) + (\alpha . \gamma)$

$(ii)\ \alpha + (\beta . \gamma) = (\alpha + \beta) . (\alpha + \gamma)$

4. Komplemen

$\forall \alpha \in B\ maka\ \exists \alpha' \in B\ sehingga\ berlaku$

$(i)\ \alpha + \alpha' = 1$

$(ii)\ \alpha . \alpha' = 0$

a	b	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

a	b	$a + b$
0	0	0
0	1	1
1	0	1
1	1	1

a	a'
0	1
1	0

***Gambar** 3: Kaidah operator biner dan operator uner*

(Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/11-Aljabar-Boolean-(2023)-bagian1.pdf, diakses pada 8 Desember 2023)

B. Fungsi Boolean

Fungsi Boolean adalah sebuah fungsi yang dibentuk oleh *n* variabel Aljabar Boolean. Representasi fungsi boolean dapat dinyatakan menggunakan 2 bentuk, yaitu Tabel Kebenaran, Rangkaian Logika, dan Aljabar.

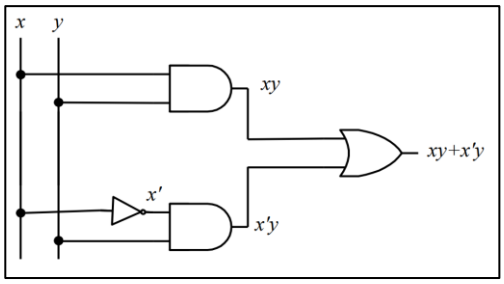
Representasi Fungsi Boolean dengan menggunakan Tabel Kebenaran adalah representasi fungsi boolean dengan menampilkan seluruh kemungkinan kombinasi variabel yang ada beserta hasil fungsinya ke dalam sebuah tabel.

x	y	z	$f(x, y, z)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

***Gambar** 4: Representasi fungsi boolean 3 variabel menggunakan Tabel Kebenaran*

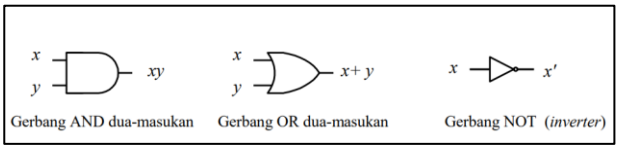
(Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/11-Aljabar-Boolean-(2023)-bagian1.pdf, diakses pada 8 Desember 2023)

Representasi Fungsi Boolean menggunakan Rangkaian Logika dilakukan dengan merepesentasikan setiap operasi pada fungsi dengan gerbang logika yang bersesuaian yang memiliki garis masukan dan keluaran. Adapun gerbang logika dasar terdapat 3 jenis, yaitu gerbang AND, gerbang OR, dan gerbang NOT.



***Gambar** 5: Representasi fungsi boolean 2 variabel menggunakan Rangkaian Logika*

(Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/11-Aljabar-Boolean-(2023)-bagian1.pdf, diakses pada 8 Desember 2023)



***Gambar** 6: Gerbang Logika Dasar*

(Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/11-Aljabar-Boolean-(2023)-bagian1.pdf, diakses pada 8 Desember 2023)

Representasi Fungsi Boolean menggunakan Aljabar sama seperti penulisan fungsi aljabar pada umumnya, dengan variabel aljabar boolean sebagai parameter atau variabel bebas dan operasi AND dan OR yang disimbolkan dengan operasi penjumlahan dan perkalian, seperti $f(x, y, z) = xz' + yz$. Representasi Fungsi Boolean menggunakan Aljabar ini dapat dispesifikasikan lebih lanjut menjadi 2 bentuk kanonik berbeda, yaitu *Product of Sum* dan *Sum of Product* dengan bentuk satu merupakan komplemen dari bentuk lainnya.

Bentuk *Product of Sum* (POS) adalah bentuk penulisan Fungsi Boolean secara Aljabar sebagai perkalian dari *maxterm*. *Maxterm* adalah suku di dalam fungsi boolean mengandung variabel yang lengkap dalam bentuk hasil jumlah dengan setiap variabel yang bernilai 0 dinyatakan tanpa komplemen, sedangkan variabel yang bernilai 1 dinyatakan menggunakan komplemen, contohnya seperti berikut.

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

$$f(x, y, z) = M_0 M_3 M_7 = \prod(0, 3, 7)$$

Bentuk *Sum of Product* (SOP) adalah bentuk penulisan Fungsi Boolean secara Aljabar sebagai penjumlahan dari *minterm*. *Minterm* adalah suku di dalam fungsi boolean mengandung variabel yang lengkap dalam bentuk hasil kali dengan setiap variabel yang bernilai 1 dinyatakan tanpa

komplemen, sedangkan variabel yang bernilai 0 dinyatakan menggunakan komplemen, contohnya seperti berikut.

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

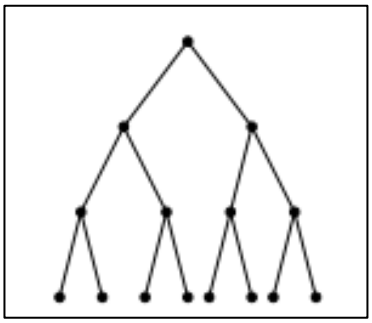
			Minterm		*Maxterm*	
x	y	z	Suku	Lambang	Suku	Lambang
0	0	0	x'y'z'	m_0	$x+y+z$	M_0
0	0	1	x'y'z	m_1	$x+y+z'$	M_1
0	1	0	x'y z'	m_2	$x+y'+z$	M_2
0	1	1	x'y z	m_3	$x+y'+z'$	M_3
1	0	0	x y'z'	m_4	$x'+y+z$	M_4
1	0	1	x y'z	m_5	$x'+y+z'$	M_5
1	1	0	x y z'	m_6	$x'+y'+z$	M_6
1	1	1	x y z	m_7	$x'+y'+z'$	M_7

***Gambar** 7: Cara membentuk minterm dan maxterm dari Tabel Kebenaran dengan 3 variabel*

(Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/11-Aljabar-Boolean-(2023)-bagian1.pdf, diakses pada 8 Desember 2023)

Fungsi Boolean dalam representasi Aljabar bentuk kanonik dapat disederhanakan menjadi memiliki lebih sedikit operasi penambahan dan perkalian tanpa mengubah nilai dari fungsi tersebut. Penyederhanaan fungsi ini dapat berguna seperti pada perancangan rangkaian digital yang memerlukan sesedikit mungkin gerbang digital untuk meingkatkan efisiensi. Terdapat 3 metode yang dapat digunakan untuk menyederhanakan Fungsi Boolean, yaitu metode Aljabar, Peta Karnaugh, dan Quine-McCluskey.

C. Pohon Biner



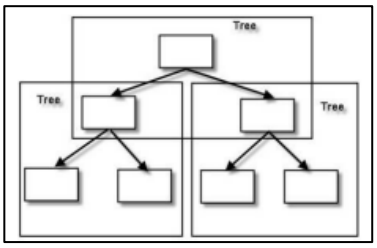
***Gambar** 8: Pohon Biner*

(Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/23-Pohon-Bag2-2023.pdf, diakses pada 8 Desember 2023)

Pohon Biner adalah graf berarah terhubung yang tidak mengandung sirkuit dan setiap simpulnya maksimal memiliki derajat keluar sejumlah dua. Simpul memiliki derajat masuk sebesar 0 adalah *Root* (akar) dari pohon biner. Pada 2 buah simpul yang terhubung, simpul dengan arah keluar disebut *parent* (orangtua) dan simpul dengan arah masuk disebut *child* (anak). Pada pohon biner, *child* dibedakan menjadi dua, yaitu *child* kiri dan *child* kanan. Dua *child* yang memiliki *parent* yang sama disebut *sibling* (saudara kandung). Setiap simpul yang ada dapat dianggap sebagai *Root* dari sebuah upapohon. Simpul yang memiliki derajat keluar sebesar 0 disebut sebagai *leaf* (daun). *Level* suatu simpul adalah banyak sisi yang dilalui pada lintasan dari *Root* menuju simpul tersebut.

Penelusuran Pohon Biner dapat dilakukan dengan 3 cara berbeda, yaitu *Preorder*, *Inorder*, dan *Postorder*. Penelusuran dengan cara *Preorder* dilakukan dengan urutan mengunjungi *Root*, mengunjungi upapohon *child* kiri secara *Preorder*, dan mengunjungi upapohon *child* kanan secara *Preorder*. Penelusuran dengan cara *Inorder* dilakukan dengan urutan mengunjungi upapohon *child* kiri secara *Inorder*, mengunjungi *Root*, dan mengunjungi upapohon *child* kanan secara *Inorder*. Penelusuran dengan cara *Postorder* dilakukan dengan urutan mengunjungi upapohon *child* kiri secara *Postorder*, mengunjungi upapohon *child* kanan secara *Postorder*, dan mengunjungi *Root*.

D. Fungsi Rekursif



***Gambar** 9:Pohon Biner sebagai Struktur yang Rekursif*

(Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/09-Rekursi-dan-relasi-rekurens-(Bagian1)-2023.pdf, diakses pada 8 Desember 2023)

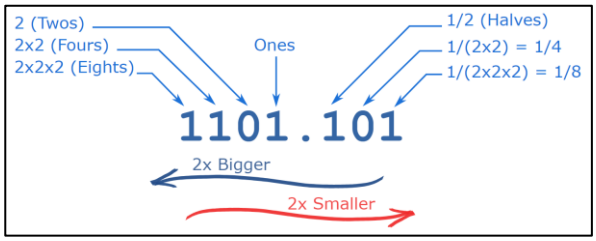
Fungsi Rekursi adalah fungsi yang terdefinisi -dalam terminologi dirinya sendiri. Fungsi Rekursi didefinisikan oleh dua bagian, yaitu Basis dan Rekurens.

Basis adalah bagian dari fungsi yang terdefinisi atas suatu nilai yang telah terdefinisi. Basis ini berguna untuk menghentikan proses rekursif. Sedangkan Rekurens adalah bagian dari fungsi yang terdefinisi dalam terminologi dirinya sendiri yang semakin mendekati basis.

E. Angka Biner

Angka Biner adalah angka yang terdiri dari digit biner (*bit*) yang hanya dapat bernilai 0 atau 1. Untuk suatu angka biner yang terdiri dari *n bit*, maka akan terdapat 2^n buah kemungkinan nilai yang dapat terbentuk.

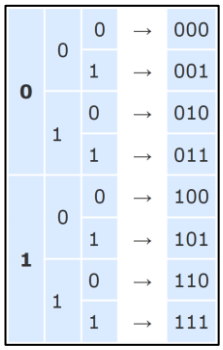
Setiap digit biner pada angka biner mewakili suatu nilai tertentu. Semakin ke kiri letak digitnya, maka nilai yang diwakili digit tersebut semakin besar 2 kali lipat, dan begitu juga sebaliknya. Hal tersebut mengakibatkan setiap digit mewakili nilai 2^n untuk *n* adalah jarak digit tersebut terhadap digit satuan.



***Gambar** 10: Representasi Nilai Setiap Digit Biner*

(Sumber:https://www.mathsisfun.com/binary-number-system.html, diakses pada 8 Desember 2023)

Angka biner memiliki suatu pola yang dapat terlihat cukup jelas. Untuk setiap angka biner dengan *n bit*, akan terdapat 2^n buah kemungkinan nilai yang dapat terbentuk, dengan 2^{n-1} buah nilai pada bagian awal pasti akan memiliki *bit* '0' pada *bit* paling kiri dan 2^{n-1} buah nilai pada bagian akhir pasti akan memiliki *bit* '1' pada *bit* paling kiri. Pada setiap bagian tersebut juga berlaku pola yang sama.



***Gambar** 11: Pola pada Angka Biner 3 bit*

(Sumber:https://www.mathsisfun.com/binary-digits.html, diakses pada 8 Desember 2023)

III. PEMBAHASAN

A. Representasi Fungsi Boolean dengan B-Trail

Suatu Fungsi Boolean dengan bentuk kanonik memiliki nilai-nilai *minterm* dan *maxterm* yang dapat merepresentasikan fungsi tersebut. Dengan memetakan nilai-nilai *minterm* dan *maxterm* tersebut ke dalam bentuk B-Trail, akan selalu menghasilkan B-Trail yang unik untuk setiap fungsi.

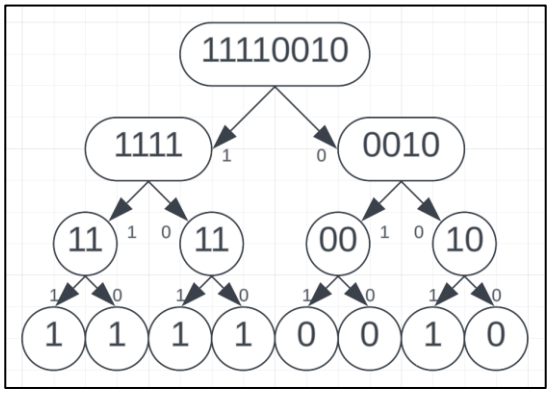
Untuk setiap sebuah *minterm* bernilai *n* pada fungsi boolean, akan direpresentasikan dengan bit '1' pada bit ke-*n* pada B-Trail. Sedangkan untuk setiap sebuah *maxterm* bernilai *n* pada fungsi boolean, akan direpresentasikan dengan bit '0' pada bit ke-*n* pada B-Trail. Panjang dari B-Trail akan sesuai dengan total *minterm* dan *maxterm* yang merepresentasikan fungsi tersebut, yaitu 2^X dengan *X* adalah banyak variabel pada fungsi. Berikut adalah contoh dari hasil representasi fungsi boolean 3 variabel dengan B-Trail.

$$f(x,y,z) = \sum(1,4,5,6,7) = \prod(0,2,3)$$
$$B-Trail\left(f(x,y,z)\right) = 11110010$$

B. Konversi B-Trail menjadi Pohon Biner

B-Trail dengan panjang 2^n merepresentasikan seluruh nilai pada *minterm* dan *maxterm* suatu fungsi, yaitu mulai dari nilai 0 sampai dengan 2^n - 1. Diketahui bahwa nilai tersebut dapat dikelompokkan menjadi dua, yaitu 2^{n-1} nilai pada bagian awal dengan *bit* paling kiri bernilai '0' dan 2^{n-1} nilai lainnya dengan *bit* paling kiri bernilai '1'. Adapun nilai pada tiap kelompok tersebut, dapat dikelompokkan dengan cara yang sama dengan menghiraukan *bit* paling kiri dari nilai tersebut.

Dengan memiliki pola tersebut, B-Trail dapat direpresentasikan menggunakan pohon biner, dengan *child* kiri adalah 2^{n-1} *bit* dari kiri dan *child* kanan adalah 2^{n-1} *bit* dari kanan. Pengelompokkan *bit* tersebut terus dilakukan hingga *Root* dari upapohon hanya memiliki 1 *bit*.



***Gambar** 12: Representasi B-Trail dalam Pohon Biner*

(Sumber: (dokumentasi pribadi), diakses pada 9 Desember 2023)

Setiap *bit* yang menjadi daun, dapat kita ketahui nilai *minterm* atau *maxterm* yang diwakili oleh *bit* tersebut dengan mencari jalur dan menggabungkan nilai dari sisi graf yang dilewati tersebut.

C. Penyederhanaan Fungsi Boolean dengan B-Trail

Langkah awal penyederhanaan Fungsi Boolean adalah dengan mengubah Fungsi Boolean ke dalam bentuk B-Trail dan merepresentasikannya ke dalam bentuk Pohon Biner. Ide utama dalam penyederhanaan Fungsi Boolean dengan B-Trail ini adalah mencari kesamaan pola yang terdapat pada kedua *child* sebuah pohon sehingga *bit* ke-*n* pada representasi nilai *minterm* atau *maxterm* dapat dihilangkan dengan *n* adalah kedalaman pohon dengan *Root* yang memiliki *child* tersebut.

Pada kali ini, penyederhaan Fungsi Boolean hanya dibatasi untuk menyederhanakan Fungsi Boolean ke dalam bentuk SOP, sehingga metode pemilihan *child* yang akan dicari kesamaan polanya dilakukan dengan mencari *child* yang merupakan saudara kandung dan memiliki kesamaan pola yang memuat paling banyak *bit* '1' yang belum pernah digunakan.

Untuk mencari kesamaan pola dalam dua buah *child* yang bersaudara kandung, dapat dilakukan dengan menerapkan operasi AND untuk tiap *bit* pada kedua *child* tersebut. Jika hasilnya sama dengan 0, itu berarti kedua *child* tersebut tidak memiliki pola yang sama. Begitu pun jika hasilnya tidak sama dengan 0, itu berarti kedua *child* tersebut memiliki pola yang sama.

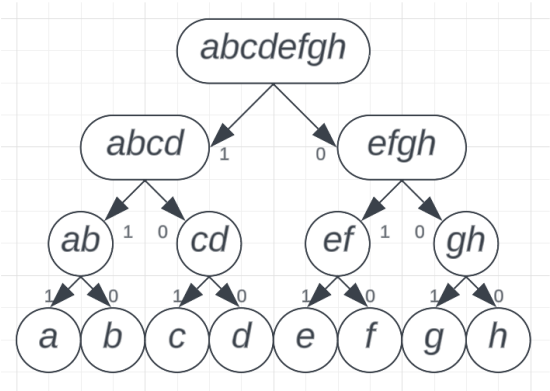
Setelah menemukan kesamaan pola tersebut, lakukan penyederhanaan ulang terhadap pola tersebut. Lalu hasilnya akan ditambahkan pada akhir string yang menunjukkan lokasi atau deretan nama sisi yang dilalui pada lintasan dari *Root* pohon menuju simpul *parent* yang memiliki kedua *child* yang memiliki kesamaan pola sebelumnya.

Sebagai gambaran lebih jelas, akan dimisalkan setiap bit dengan suatu variabel aljabar. Pada contoh kali ini, akan digunakan Fungsi Boolean dengan 3 variabel sehingga diperoleh B-Trail dengan panjang 2^3, *'abcdefgh'*, dengan

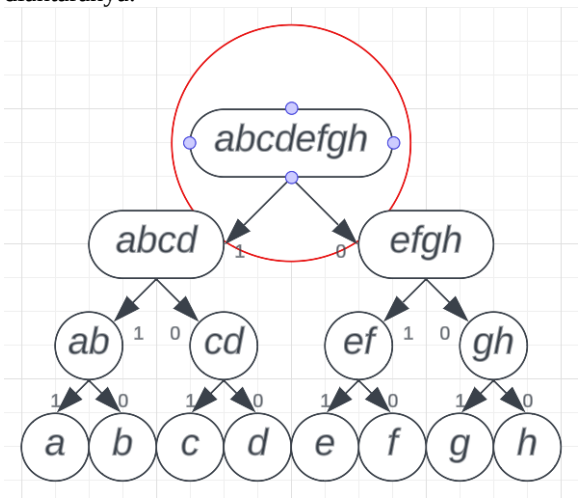
berikut adalah langkah penyederhanaan yang dilakukan.

1. Konversi B-Trail menjadi Pohon Biner.

$$B - Trail\ (f(x,y,z)) = abcdefgh$$



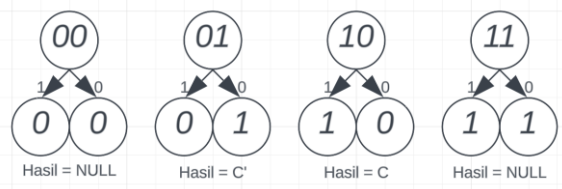
2. Memilih simpul dengan panjang nilai lebih dari sama dengan 2 dan kemungkinan hasil AND kedua *child*-nya memiliki *bit* '1' paling banyak yang belum pernah digunakan dan memiliki panjang nilai terpendek diantaranya.



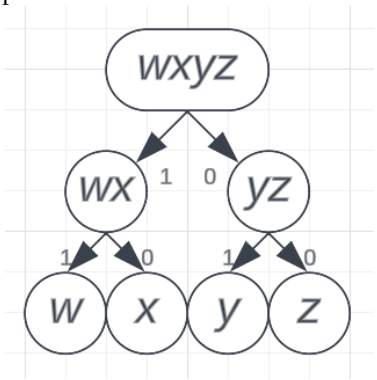
3. Mendapatkan kesamaan pola kedua *child* dalam bentuk B-Trail dan menandai setiap *bit* '1' pada pola tersebut sebagai 'telah digunakan'.

$$\begin{array}{l} abcd \\ \underline{efgh\ \&} \\ wxyz \end{array} \quad , dengan\ wxyz \neq 0$$

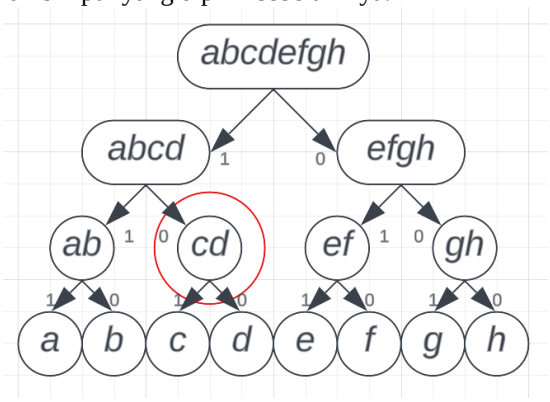
4. Jika panjang B-Trail dari pola tersebut sama dengan 1, maka hasil penyederhanaannya adalah *P dengan P* bernilai NULL jika kedua *child* memiliki nilai yang sama, bernilai nama variabel terakhir jika hanya *child* dengan sisi '1' yang memiliki *bit* '1', dan bernilai komplemen variabel terakhir jika hanya *child* dengan sisi '0' yang memiliki *bit* '1'.



5. Jika panjang B-Trail dari pola tersebut lebih dari 2, maka ulangi langkah penyederhanaan dari tahap 1 dan seterusnya dengan menggunakan B-Trail dari pola yang didapatkan.



6. Hasil penyederhanaan adalah string deretan nilai sisi pada lintasan yang telah dikonversi dengan kententuan sisi bernilai '1' dikonversi menjadi variabel boolean yang ke-*n* dari kanan dan sisi bernilai '0' dikonversi menjadi komplemen variabel boolean yang ke-*n* dengan *n* adalah level dari simpul yang dilalui dari *Root* menuju simpul yang dipilih sebelumnya, kemudian dilanjutkan dengan hasil penyederhanaan pada nilai B-Trail simpul yang dipilih sebelumnya.



$$f(abcdefgh) = AB' + f(cd)$$

7. Ulangi kembali langkah 1 dengan menggunakan B-Trail utama fungsi boolean hingga seluruh *bit* '1' pada B-Trail utama fungsi boolean 'telah digunakan'.

$'misal$:

$hasil\ penyederhanaan\ 1: A$
$hasil\ penyederhanaan\ 2: AB'$
$hasil\ penyederhanaan\ 3: BC'$

8. Gabungkan seluruh hasil penyederhanaan yang didapatkan dengan dipisahkan oleh operasi '+'.

$$f(abcdefgh) = A + AB' + BC'$$

D. Analisis Kode Program 'Penyederhanaan Fungsi Boolean dengan B-Trail'

Dalam pembuatan kode program, terdapat beberapa penyesuaian dalam algoritma penyederhanaan fungsi boolean dengan B-Trail dengan tujuan untuk meningkatkan efisiensi dan efektifitas kode program.

Representasi B-Trail menggunakan pohon biner dilakukan dengan memanfaatkan *multidimensional* array, dengan setiap array hanya memiliki 2 buah elemen yang dapat berupa array atau karakter dan setiap elemen array tersebut

merepresentasikan sebuah simpul pada pohon. Oleh karena itu, dibutuhkan juga fungsi konversi dari *multidimensional* array menjadi string untuk mendapatkan nilai pada simpul tersebut yang pada kode ini adalah fungsi 'listToString(list)'. Contoh penerapan *multidimensional* array pada representasi pohon biner B-Trail adalah sebagai berikut.

$$B - Trail = 11110010$$

$$Pohon\ Biner = \Big[\big[[[1][1]]\ [[1][1]]\big]\ \big[[[0][0]]\ [[1][0]]\big]\Big]$$

Pada B-Trail, normalnya *bit* yang merepresentasikan literal dengan nilai 0 adalah *bit* yang paling kanan. Akan tetapi, pada pengaplikasiannya di dalam kode program, letak *bit* yang merepresentasikan literal dengan nilai 0 berada di paling kiri. Semakin kanan *bit* pada B-Trail, maka akan merepresentasikan nilai literal yang lebih besar. Hal tersebut dilakukan untuk menyesuaikan index 0 pada array yang berada pada paling kiri sehingga index array tersebut dapat langsung merepresentasikan nilai *bit* pada literal tersebut. Contohnya adalah sebagai berikut.

$$B - Trail\ Normal = 11110010$$

$$B - Trail\ Program = 01001111$$

Selama proses penyederhanaan, terdapat perbedaan pada metode penyimpanan literal hasil penyederhanaan pada kode program. Jika pada metode sebelumnya, hasil penyederhanaan langsung ditulis menggunakan nama variabel pada fungsi boolean. Sedangkan pada kode program, hasil penyederhanaan tetap ditulis dalam *bit* dan untuk variabel yang dihilangkan dapat dimisalkan menjadi karakter 'X'. Hal tersebut dilakukan agar dapat dilakukan konversi lebih lanjut dari hasil yang masih berbentuk biner ke dalam representasi variabel booleannya dengan karakter 'X' yang menandakan bahwa variabel tersebut dapat dihilangkan. Contohnya adalah sebagai berikut.

$$Hasil = 1X0 = AC'$$

$$Hasil = XX1 = C$$

$$Hasil = X01 = B'C$$

```
''' MAIN PROGRAM '''
nVar = int(input("Masukkan banyak variabel: "))
input = input("Masukkan bentuk SOP: ")

# inisiasi B-trail
Btrail = ['0' for _ in range(2**nVar)]
nonUsedBtrail = ['0' for _ in range(2**nVar)]
listHasil = []

# membuat B-trail berdasarkan input
input = input.split(" ")
for i in input:
    Btrail[int(i)] = '1'
    nonUsedBtrail[int(i)] = '1'

Btrail = "".join(Btrail)
print('B-TRAIL: ' + Btrail)
Btrail = splitList(Btrail) # convert B-Trail menjadi Pohon Biner dalam bentuk array

# Kasus Semesta atau Null
if (int(listToString(Btrail)) == 0): # Kondisi: seluruh bit bernilai 0
    print('Null')
elif (int(listToString(Btrail), 2) == (2**(2**nVar)-1)): # Kondisi: seluruh bit bernilai 1
    print('All True')
else:
    listHasil = []
    searchPattern(Btrail, "", False)
    print(f'LIST_HASIL: ', end='')
    print(listHasil)
    parsingHasil()
```

***Gambar** 13: Main Program Penyederhanaan Aljabar Boolean dengan B-Trail*

(Sumber: (dokumentasi pribadi), diakses pada 10 Desember 2023)

Pada kode main program di *Gambar 13*, kode akan meminta input dari pengguna terkait informasi jumlah variabel yang ada pada fungsi boolean dan juga informasi bentuk SOP fungsi boolean tersebut. Dengan informasi jumlah variabel, diinisiasi sebuah B-Trail kosong dalam sebuah list 'Btrail' dengan panjang 2^n dengan *n* adalah jumlah variabel. Dengan informasi representasi SOP fungsi boolean, dapat dilakukan pengubahan *bit* ke-*x* dengan *x* adalah nilai-nilai *minterm* pada representasi SOP. Lalu, diinisiasi juga sebuah list bernama 'nonUsedBtrail' dengan cara yang sama untuk mengetahui *bit* '1' mana yang telah dikategorikan sebagai 'telah digunakan'. List 'Btrail' yang telah dibuat sebelumnya, dikonversi menjadi pohon biner menggunakan fungsi 'splitList(Btrail)'. Selanjutnya merupakan perintah untuk memulai penyederhanaan dari pohon biner B-Trail yang ada.

```
def searchPattern(list, prevIdxStr, isDoneComparing):
    global nonUsedBtrail
    global listHasil

    string0 = listToString(list[0])
    string1 = listToString(list[1])

    print('\nHASIL NOW: '+prevIdxStr)
    print('STRING0: '+string0)
    print('STRING1: '+string1+'\n')

    stop = False # bernilai True saat telah mencapai kondisi basis

    if (len(prevIdxStr) == nVar-1): # mengecek apakah sudah mencapai kondisi basis
        stop = True

    andOp = int(string0, 2) & int(string1, 2) # persamaan pola antara kedua child

    if (andOp != 0): # Kondisi: kedua child tidak memiliki pola
        # Kondisi: pola kedua child memuat bit '1' yang belum digunakan
        if (isDoneComparing or ((geserString(andOp, prevIdxStr+'1') & int("".join(nonUsedBtrail), 2)) != 0)
        or ((geserString(andOp, prevIdxStr+'0') & int("".join(nonUsedBtrail), 2)) != 0)):
            isDoneComparing = True # bernilai True saat tidak perlu dicek apakah bit '1' yang ada telah digunakan
            panjang = len(string0)

            print('Menambahkan X!')
            if (not stop): # Kondisi Basis
                listBaru = splitList(paddingBin(andOp, panjang))
                searchPattern(listBaru, prevIdxStr+'X', isDoneComparing) # menyederhanakan B-Trail pola yang didapat
            else: # Kondisi Rekurens
                deleteByStrIdx(prevIdxStr+'X') # menandai bit '1' sebagai 'telah digunakan'
                print('LIST NONUSED: ', end='')
                print(nonUsedBtrail)

            isDoneComparing = False

    if (int("".join(nonUsedBtrail),2) != 0): # Kondisi: masih terdapat bit '1' yang belum digunakan
        cmp0 = int(string0, 2) != 0
        cmp1 = int(string1, 2) != 0
        if (cmp0 and cmp1): # Kondisi: child kiri & kanan tidak bernilai nol dan tidak memiliki pola
            isCompare2 = False # bernilai True saat tidak perlu dicek apakah bit '1' yang ada telah digunakan
        else:
            isCompare2 = isDoneComparing
        if (cmp0): # Kondisi: child kiri tidak bernilai nol
            if (isCompare2 or ((geserString(int(string0, 2), prevIdxStr+'0') & int("".join(nonUsedBtrail), 2)) != 0)):
                print('Menambahkan 0!')
                if (not stop): # Kondisi Rekurens
                    searchPattern(list[0], prevIdxStr+"0", isDoneComparing) # menulusuri child kiri
                else: # Kondisi Basis
                    deleteByStrIdx(prevIdxStr+'0') # menandai bit '1' sebagai 'telah digunakan'
                    print('LIST NONUSED: ', end='')
                    print(nonUsedBtrail)

        if (cmp1): # Kondisi: child kanan tidak bernilai nol
            if (isCompare2 or ((geserString(int(string1, 2), prevIdxStr+'1') & int("".join(nonUsedBtrail), 2)) != 0)):
                print('Menambahkan 1!')
                if (not stop): # Kondisi Rkurens
                    searchPattern(list[1], prevIdxStr+'1', isDoneComparing) # menulusuri child kanan
                else: # Kondisi Basis
                    deleteByStrIdx(prevIdxStr+'1') # menandai bit '1' sebagai 'telah digunakan'
                    print('LIST NONUSED: ', end='')
                    print(nonUsedBtrail)
```

***Gambar** 14: Fungsi Penyederhanaan Fungsi Boolean dengan B-Trail*

(Sumber: (dokumentasi pribadi), diakses pada 10 Desember 2023)

Pada kode fungsi penyederhaan Fungsi Boolean dengan B-Trail di *Gambar 14*, dilakukan pengaksesan variabel 'listHasil' dan 'nonUsedBtrail' secara global karena kedua variabel tersebut perlu untuk diubah dan diperbaharui pada saat proses rekursi sedang berjalan. Lalu, diinisiasi juga sebuah variabel 'stop' yang akan bernilai True jika sudah mencapai kondisi basis, yaitu saat panjang hasil penyederhanaan sama dengan banyak variabel pada fungsi boolean.

Fungsi penyederhanaan tersebut menggunakan metode penelusuran *Preorder*, sehingga peletakan kondisi untuk

mengecek pola pada kedua *child* pada *Root* diletakkan paling awal dan kondisi selanjutnya diletakkan dibawahnya dengan menggunakan kondisi '*if'* yang terpisah karena penelusuran secara *Preorder* tidak tergantung pada kondisi apapun dan tetap mengecek sesuai urutannya.

Pada saat proses penyederhanaan, terkadang perlu diperhatikan apakah seluruh *bit* '1' yang digunakan 'telah digunakan' agar tidak terjadi penyederhanaan ganda terhadap kelompok literal yang sebenarnya sama. Akan tetapi, terdapat juga kasus saat tidak perlu diperhatikan apakah seluruh *bit* '1' yang digunakan 'telah digunakan', yaitu pada saat pola yang dihasilkan oleh kedua *child* mengandung beberapa *bit* '1' yang 'telah digunakan' namun tidak seluruhnya. Sehingga saat dilakukan rekursif dan hingga dilakukannya pencarian pola pada *bit* '1' yang 'telah digunakan' tersebut, maka hal tersebut seharusnya proses tersebut tetap boleh dilanjutkan. Oleh karena itu, dibuatlah variabel 'isDoneComparing' yang bernilai True saat memulai untuk melakukan penyederhanaan terhadap pola pada kedua *child* yang menandakan bahwa *bit* '1' yang ada tidak perlu dicek lagi apakah 'telah digunakan' atau tidak. Namun, terdapat kasus khusus lain, yaitu pada saat pola yang dihasilkan direpresentasikan sebagai *Root* pada pohon biner yang baru dan kedua *child*-nya tidak memiliki pola, maka akan terjadi percabangan yang akan menghasilkan 2 buah hasil literal. Oleh karena itu, pada kasus tersebut perlu dilakukan pengecekan apakah *bit* '1' yang ada pada masing-masing *child* 'telah digunakan' atau tidak, sehingga dibuatlah variabel isCompare2 yang akan selalu bernilai False saat terjadi kasus percabangan yang menandakan bahwa perlu dilakukan pengecekan apakah seluruh *bit* '1' yang ada 'telah digunakan' atau tidak.

Pada kondisi rekurens fungsi tersebut, diperhatikan parameter 'prevIdxStr' yang menyimpan sisi-sisi yang telah dilewati sebelumnya. Pada kondisi rekurens, dilakukan pemanggilan terhadap dirinya sendiri dengan menambahkan nilai dari sisi yang dilalui dari *Root* menuju simpul selanjutnya yang memiliki nilai tak nol pada parameter 'prevIdxStr'. Jika terdapat pola pada kedua *child*, maka akan ditambahkan karakter 'X' pada parameter 'prevIdxStr' yang menandakan bahwa variabel tersebut dapat diabaikan.

Pada kondisi basis, dilakukan penambahan hasil yang telah didapatkan kedalam 'listHasil' dan dilakukan juga pengkategorian *bit* '1' yang ada menjadi 'telah digunakan' dengan mengubah *bit* '1' yang digunakan menjadi *bit* '0' pada 'nonUsedBtrail' dengan menggunakan fungsi 'deleteByStrIdx(strIdx)'.

Fungsi penyederhanaan fungsi boolean dengan B-Trail tersebut akan berhenti saat setiap simpul telah ditelurusi dengan metode *Preorder*. Lalu akan dilakukan *parsing* terhadap 'listHasil' yang ada dengan mengganti *bit* ke-*n* dengan variabel ke-*n* dan menggabungkan seluruh literal tersebut dengan operasi '+' diantaranya.

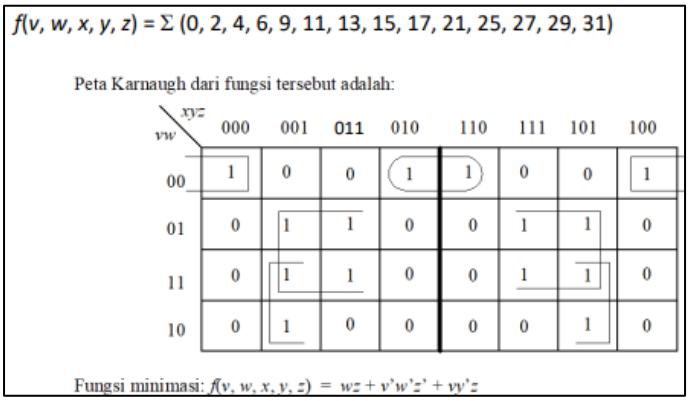
```
def splitList(string): # mengubah string menjadi list biner
    panjang = len(string)
    if (panjang == 1):
        return [string]
    else:
        return [splitList(string[:panjang//2]), splitList(string[panjang//2:])]

def listToString(list): # mengubah list biner menjadi string
    panjang = len(list)
    if (panjang == 1):
        return str(list[0])
    else:
        return str(listToString(list[0]) + listToString(list[1]))

def paddingBin(biner, pad): # membuat string biner dengan panjang pad
    strings = bin(biner)[2:]
    totalpad = pad - len(strings)
    strings = totalpad*'0' + strings

    return strings

def deleteByStrIdx(strIdx):
    global nonUsedBtrail
    print('\n----- Menambahkan', strIdx, 'ke Hasil Utama! -----\n')
    listHasil.append(strIdx)
    strIdxList = convertXtoValidStr([strIdx])

    for i in strIdxList:
        idx = int(i, 2)
        nonUsedBtrail[idx] = '0'

def geserString(intString, strIdx):
    i = nVar

    for j in strIdx:
        i -= 1
        if (j == '0'):
            intString <<= 2**i

    return intString

def convertXtoValidStr(strIdxList):
    panjang = len(strIdxList)
    ketemuX = False

    for _ in range(panjang):
        a = strIdxList.pop(0)
        idx = a.find('X')
        if (idx != -1):
            ketemuX = True
            strIdxList.append(a[:idx] + '0' + a[idx+1:])
            strIdxList.append(a[:idx] + '1' + a[idx+1:])
        else:
            strIdxList.append(a)

    if (ketemuX):
        strIdxList = convertXtoValidStr(strIdxList)

    return strIdxList

def parsingHasil():
    global listHasil
    for urutan in range(len(listHasil)):
        hasil = listHasil[urutan]
        for i in range(len(hasil)):
            if (hasil[i] == '1'):
                print(chr(65+i), end='')
            elif (hasil[i] == '0'):
                print(chr(65+i)+ '\'', end='')
        if (urutan != len(listHasil)-1):
            print(' + ', end='')
```

***Gambar** 15: Fungsi-Fungsi Bantuan yang Digunakan pada Kode Program*

(Sumber: (dokumentasi pribadi), diakses pada 10 Desember 2023)

D. Pengujian Kode Program 'Penyederhanaan Fungsi Boolean dengan B-Trail'

Pada pengujian kode program, akan dibandingkan hasil penyederhanaan dengan Peta Karnaugh dan hasil penyederhanaan dengan Kode Program 'Penyederhanaan Fungsi Boolean dengan B-Trail'.



***Gambar** 16: Hasil Penyederhanaan dengan Peta Karnaugh 5 variabel*

(Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/12-Aljabar-Boolean-(2023)-bagian2.pdf, diakses pada 10 Desember 2023)

```
Masukkan banyak variabel: 5
Masukkan bentuk SOP: 0 2 4 6 9 11 13 15 17 21 25 27 29 31
B-TRAIL: 10101010010101010100010001010101

HASIL NOW:
STRING0: 1010101001010101
STRING1: 0100010001010101

Menambahkan X!

HASIL NOW: X
STRING0: 00000000
STRING1: 01010101

Menambahkan 1!

HASIL NOW: X1
STRING0: 0101
STRING1: 0101

Menambahkan X!

HASIL NOW: X1X
STRING0: 01
STRING1: 01

Menambahkan X!

HASIL NOW: X1XX
STRING0: 0
STRING1: 1

Menambahkan 1!

----- Menambahkan X1XX1 ke Hasil Utama! -----

LIST NONUSED: 10101010000000000100010000000000
Menambahkan 0!

HASIL NOW: 0
STRING0: 10101010
STRING1: 01010101

Menambahkan 0!

HASIL NOW: 00
STRING0: 1010
STRING1: 1010

Menambahkan X!

HASIL NOW: 00X
STRING0: 10
STRING1: 10

Menambahkan X!

HASIL NOW: 00XX
STRING0: 1
STRING1: 0

Menambahkan 0!

----- Menambahkan 00XX0 ke Hasil Utama! -----
```

```
LIST NONUSED: 00000000000000000100010000000000
Menambahkan 1!

HASIL NOW: 1
STRING0: 01000100
STRING1: 01010101

Menambahkan X!

HASIL NOW: 1X
STRING0: 0100
STRING1: 0100

Menambahkan X!

HASIL NOW: 1XX
STRING0: 01
STRING1: 00

Menambahkan 0!

HASIL NOW: 1XX0
STRING0: 0
STRING1: 1

Menambahkan 1!

----- Menambahkan 1XX01 ke Hasil Utama! -----

LIST NONUSED: 00000000000000000000000000000000
LIST_HASIL: ['X1XX1', '00XX0', '1XX01']
BE + A'B'E' + AD'E
```

***Gambar** 17: Hasil Pengujian Kode Program 'Penyederhanaan Fungsi Boolean dengan B-Trail' 5 variabel*

(Sumber: (dokumentasi pribadi), diakses pada 10 Desember 2023)

Pada *Gambar 16* dan *17*, dapat dilihat bahwa hasil pengujian kode program 'Penyederhanaan Fungsi Boolean dengan B-Trail' memiliki hasil penyederhanaan yang sama dengan hasil penyederhanaan dengan Peta Karnaugh. Hal tersebut dapat menjadi salah satu bukti bahwa kode program sekaligus metode penyederhaan dengan B-Trail dapat berjalan dengan baik.

```
Masukkan banyak variabel: 7
Masukkan bentuk SOP: 127 126 63 62 64 65 66 67 0 1 2 3
B'C'D'E' + BCDEF
```

Gambar 18: Hasil Pengujian Kode Program 'Penyederhanaan Fungsi Boolean dengan B-Trail' 7 variabel

(Sumber: (dokumentasi pribadi), diakses pada 10 Desember 2023)

Pada *Gambar 18*, diambil sebuah contoh fungsi sederhana dan dapat diketahui bahwa hasil penyederhanaan fungsi boolean tersebut adalah paling sederhana. Hal tersebut dapat membuktikan juga bahwa Kode Program tersebut dapat menyederhanakan fungsi boolean dengan jumlah variabel berapapun.

IV. KESIMPULAN

Aplikasi Fungsi Rekursif dan Pohon Biner dalam Algoritma Penyederhanaan Fungsi Boolean Menggunakan B-Trail dapat disimpulkan sebagai berikut.

1. Fungsi Rekursif dan Pohon Biner dapat diterapkan pada Algoritma Penyederhanaan Fungsi Boolean dengan B-Trail.

2. Algoritma Penyederhanaan Fungsi Boolean dengan B-Trail dapat menghasilkan nilai atau hasil penyederhanaan yang sama jika dibandingkan dengan Penyederhanaan Fungsi Boolean dengan Peta Karnaugh.
3. Algoritma Penyederhanaan Fungsi Boolean dengan B-Trail dapat diimplementasikan dalam bentuk kode program sehingga memiliki kecepatan penyederhanaan yang cepat.
4. Algoritma Penyederhanaan Fungsi Boolean dengan B-Trail dapat menyederhanakan fungsi boolean dengan jumlah variabel berapapun.

V. UCAPAN TERIMA KASIH

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa yang telah memberikan karunia, sehingga penulis dapat Makalah IF2120 Matematika Diskrit – Sem. I Tahun 2023/2024 menyelesaikan makalah yang berjudul "Aplikasi Fungsi Rekursif dan Pohon Biner dalam Algoritma Penyederhanaan Fungsi Boolean Menggunakan B-Trail" yang dapat selesai tepat pada waktunya. Tak lupa juga penulis mengucapkan terima kasih kepada Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc., Ibu Dr. Fariska Zakhralativa Ruskanda, S.T., M.T., dan Bapak Dr. Ir. Rinaldi, M.T yang telah membimbing kami pada mata kuliah IF2120 Matematika Diskrit. Terakhir, penulis mengucapkan terima kasih kepada orang tua, keluarga, dan seluruh pihak yang membantu penulis dalam menyelesaikan makalah ini.

REFERENCES

[1] Rinaldi Munir, "Aljabar Boolean (Bag. 1)", https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/Aljabar-Boolean-(2023)-bagian1.pdf, diakses 9 Desember 2023.
[2] Rinaldi Munir, "Aljabar Boolean (Bag. 2)", https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/12-Aljabar-Boolean-(2023)-bagian2.pdf, diakses 9 Desember 2023.
[3] Rinaldi Munir, "Pohon (Bag. 2)", https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/23-Pohon-Bag2-2023.pdf, diakses 9 Desember 2023.
[4] Rinaldi Munir, "Rekursi dan Relasi Rekurens Bagian 1", https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/09-Rekursi-dan-relasi-rekurens-(Bagian1)-2023.pdf, diakses 9 Desember 2023.
[5] Rod Pierce, "Binary Number System", https://www.mathsisfun.com/binary-number-system.html, diakses pada 9 Desember 2023 .
[6] Rod Pierce, "Binary Digits", https://www.mathsisfun.com/binary-digits.html, diakses 9 Desember 2023.

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, 11 Desember 2023

Ahmad Hasan Albana (13522041)