Logika Matematika untuk Rekayasa Perangkat Lunak Modul Belajar dengan Visual yang Dapat Dibaca Tanpa Melihat

Muhammad Riduwan Khafidi
Program Studi Teknik Rekayasa Perangkat Lunak
Politeknik Negeri Batam
4342611034@students.polibatam.ac.id

November 5, 2026

Abstrak

Logika proposisi adalah bahasa formal yang membuat komputer dapat mengambil keputusan berdasarkan aturan yang pasti. Modul ini membahas definisi proposisi, lima operator dasar beserta tabel kebenarannya, cara membangun tabel kebenaran majemuk, klasifikasi ekspresi menjadi tautologi, kontradiksi, atau kontingensi, hukum-hukum logika, ekivalensi, De Morgan, dan penyederhanaan ekspresi. Setiap konsep dibuka dengan analogi konkret, dilengkapi contoh berangka dari perangkat lunak, dan ditutup dengan tabel ringkas. Sepuluh visual dirancang hitam-putih dan disertai narasi pengganti gambar sehingga pembaca yang mengandalkan teks tetap dapat memahaminya. Modul ini memuat dua studi kasus yang dikerjakan penuh: validasi akses dashboard admin dengan enam belas baris tabel kebenaran, dan validasi formulir peminjaman buku perpustakaan. Setelah membaca modul ini, pembaca dapat menuliskan aturan keputusan program, membangun tabel kebenarannya, membuktikan bahwa aturan itu benar untuk semua kombinasi kondisi, dan menyederhanakan ekspresi yang tampak rumit menjadi bentuk yang lebih mudah dibaca dan diuji.

1 Pendahuluan

1.1 Analogi Pembuka: Penjaga Pintu di Gedung Kantor

Bayangkan sebuah gedung kantor dengan seorang penjaga pintu di depan pintu masuk ruang server. Penjaga itu punya empat syarat yang harus dipenuhi sebelum pintu dibuka. Pertama, pengunjung harus punya kartu identitas yang sah. Kedua, nama pengunjung harus terdaftar di daftar izin hari itu. Ketiga, pengunjung harus memasukkan kode akses yang benar. Keempat, pengunjung tidak sedang dalam daftar blokir.

Jika satu syarat saja gagal, pintu tetap tertutup. Penjaga tidak memutuskan berdasarkan perasaan. Penjaga memutuskan berdasarkan aturan yang sudah ditulis dan diperiksa untuk setiap kemungkinan.

Komputer bekerja seperti penjaga pintu itu. Ia hanya mengenal benar atau salah, satu atau nol.

Tidak ada "mungkin", "kira-kira", atau "biasanya". Setiap keputusan yang rumit harus dipecah menjadi proposisi-proposisi sederhana, lalu digabungkan dengan operator logika yang pasti.

1.2 Mengapa Topik Ini Penting

Setiap pernyataan `if`, `while`, dan `switch` di program adalah aturan logika. Setiap validasi formulir, setiap pemeriksaan hak akses, setiap syarat diskon adalah ekspresi logika. Jika ekspresi itu salah tulis, program akan berjalan normal pada kebanyakan kasus, tetapi gagal pada kasus yang jarang. Bug seperti ini paling sulit ditemukan karena muncul hanya pada kombinasi kondisi tertentu.

Tabel kebenaran adalah alat yang memaksa kita memeriksa semua kombinasi, bukan hanya kombinasi yang terpikir saat menulis kode. Jika sebuah aturan punya empat variabel, ada 16 kombinasi. Jika tabel kebenaran menunjukkan aturan berperilaku benar untuk 16 kombinasi, aturan itu terbukti benar untuk semua kondisi [**?**].

1.3 Cara Membaca Modul Ini

Setiap bagian mengikuti pola yang sama. Pertama, analogi konkret. Kedua, definisi formal. Ketiga, tabel kebenaran atau contoh berangka. Keempat, visual dengan narasi pengganti gambar. Kelima, padanan di perangkat lunak. Setiap konsep kunci diberi dua bagian tambahan: Cara Berpikir yang menjelaskan langkah mental menyelesaikan soal, dan Kalau Salah yang menunjukkan akibat jika langkah dilewati.

1.4 Peta Pembahasan

Bagian kedua membahas proposisi. Bagian ketiga membahas lima operator dasar. Bagian keempat membahas cara membangun tabel kebenaran majemuk. Bagian kelima membahas klasifikasi ekspresi. Bagian keenam membahas hukum-hukum logika. Bagian ketujuh membahas ekivalensi dan De Morgan. Bagian kedelapan membahas penyederhanaan. Bagian kesembilan dan kesepuluh memuat dua studi kasus. Bagian kesebelas dan keduabelas memberi daftar kesalahan umum dan lembar rujukan cepat. Bagian ketigabelas menutup dengan tiga prinsip.

2 Proposisi

2.1 Analogi: Kalimat yang Bisa Diadili

Bayangkan seorang hakim yang hanya mau mendengar kalimat yang bisa dinilai benar atau salah. Kalimat "Ibu kota Indonesia adalah Jakarta" bisa dinilai, dan hakim bisa memutuskan bahwa itu benar. Kalimat "Tolong tutup pintunya" tidak bisa dinilai, karena itu perintah, bukan pernyataan. Kalimat "x lebih besar dari 5" juga tidak bisa dinilai, karena nilai x belum diketahui.

Proposisi adalah kalimat yang bisa diadili hakim itu. Ia harus punya nilai benar atau salah, tidak keduanya, dan tidak mungkin tidak punya nilai.

2.2 Definisi Formal

Proposisi adalah pernyataan yang bernilai benar atau salah, tetapi tidak keduanya. Nilai kebenaran ditulis B untuk benar dan S untuk salah, atau dalam bahasa Inggris T untuk true dan F untuk false [**?**].

Sebuah pernyataan disebut proposisi jika memenuhi tiga syarat. Pertama, ia adalah kalimat berita, bukan pertanyaan, perintah, atau seruan. Kedua, nilainya pasti, tidak berubah-ubah. Ketiga, ia tidak mengandung variabel bebas yang belum diberi nilai.

Contoh proposisi: "3+2=5" bernilai benar. "Polibatam ada di Batam" bernilai benar. "1+1=3" bernilai salah. Contoh bukan proposisi: "x>5" karena x belum diberi nilai, "Tolong tutup pintunya" karena itu perintah, dan "Berapa harga tiket?" karena itu pertanyaan.

2.3 Proposisi di Perangkat Lunak

Dalam program, proposisi adalah ekspresi boolean. Setiap kondisi di dalam `if`, `while`, atau `switch` adalah proposisi yang sudah diberi nilai. Contoh: `status_login == true`, `saldo > 0`, `umur >= 17`, dan `isAdmin == true`.

Pemrogram harus memastikan setiap ekspresi boolean benar-benar punya nilai. Jika ada variabel yang belum diinisialisasi, nilainya tidak pasti, dan ekspresi itu bukan proposisi. Program yang bekerja dengan ekspresi bukan proposisi berperilaku tidak dapat diprediksi. Perhatikan Gambar **??**.

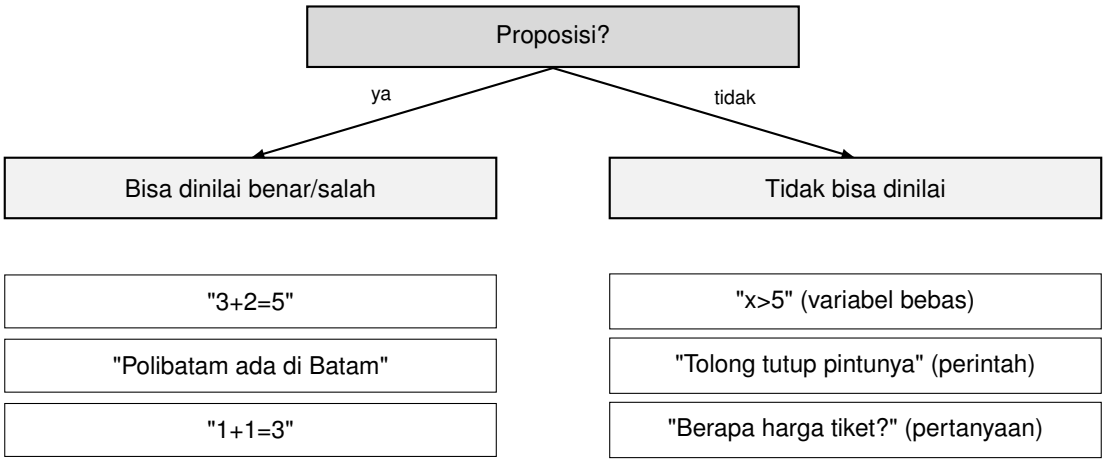


Figure 1. Diagram alur memeriksa apakah sebuah pernyataan adalah proposisi.

Cara membaca Gambar **??**: gambar ini menampilkan satu kotak di atas bertuliskan "Proposisi?" dengan dua cabang panah ke bawah. Cabang kiri menuju kotak "Bisa dinilai benar/salah", cabang kanan menuju kotak "Tidak bisa dinilai". Di bawah cabang kiri ada tiga contoh pernyataan yang memenuhi syarat: "3+2=5", "Polibatam ada di Batam", dan "1+1=3". Di bawah cabang

kanan ada tiga contoh pernyataan yang tidak memenuhi syarat: "x>5" karena ada variabel bebas, "Tolong tutup pintunya" karena perintah, dan "Berapa harga tiket?" karena pertanyaan. Kotak paling atas diberi latar abu-abu gelap untuk menandakan posisi keputusan. Inti pesan gambar ini adalah memeriksa proposisi hanya butuh satu pertanyaan: apakah kalimat ini bisa diputuskan benar atau salahnya tanpa informasi tambahan.

2.4 Cara Berpikir: Memeriksa Proposisi

Untuk memeriksa apakah sebuah kalimat adalah proposisi, ajukan dua pertanyaan berurutan. Pertama, apakah kalimat ini kalimat berita? Jika perintah, pertanyaan, atau seruan, bukan proposisi. Kedua, apakah semua variabel di dalamnya sudah diberi nilai? Jika ada variabel bebas, bukan proposisi sampai variabel itu diberi nilai.

Jika kedua pertanyaan itu dijawab, kalimat itu adalah proposisi. Jika salah satu dilanggar, kalimat itu bukan proposisi.

3 Lima Operator Dasar

3.1 Analogi Setiap Operator

Setiap operator logika punya padanan di dunia nyata. Negasi adalah tombol lampu yang ditekan sekali untuk membalik keadaan. Konjungsi adalah dua kunci yang harus dibuka bersama untuk membuka peti. Disjungsi adalah dua pintu yang salah satunya cukup dibuka untuk masuk. Implikasi adalah janji bersyarat, misalnya "kalau kamu lulus ujian, saya beri hadiah". Biimplikasi adalah lampu yang dirangkai langsung dengan sakelarnya, sehingga menyala tepat ketika sakelar ditekan dan mati tepat ketika sakelar dilepas.

3.2 Negasi: $\neg A$

Negasi membalik nilai kebenaran. Jika A benar, $\neg A$ salah. Jika A salah, $\neg A$ benar. Tabel kebenarannya sangat pendek, hanya dua baris, ditampilkan pada Tabel **??** [**?**].

Table 1. Tabel kebenaran negasi.

A	$\neg A$
B	S
S	B

Di perangkat lunak, negasi dipakai untuk membalik kondisi. Contoh: A adalah "akun sedang diblokir", maka $\neg A$ adalah "akun tidak sedang diblokir". Di kode: `if (!isBlocked) { ... }`. Tanda seru adalah operator negasi di hampir semua bahasa pemrograman.

3.3 Konjungsi: $A \wedge B$

Konjungsi bernilai benar hanya jika A dan B keduanya benar. Jika salah satu salah, hasilnya salah. Tabel kebenarannya ditampilkan pada Tabel **??**.

Table 2. Tabel kebenaran konjungsi.

A	B	$A \wedge B$
B	B	B
B	S	S
S	B	S
S	S	S

Di perangkat lunak, konjungsi dipakai untuk syarat yang harus dipenuhi semua. Contoh: L adalah "login valid" dan A adalah "berperan sebagai admin", maka $L \wedge A$ adalah "login valid dan berperan admin". Di kode: `if (isLoggedIn && isAdmin) { tampilkanDashboard(); }`.

3.4 Disjungsi: $A \vee B$

Disjungsi bernilai benar jika salah satu dari A atau B benar, termasuk jika keduanya benar. Disjungsi hanya bernilai salah jika keduanya salah. Tabel kebenarannya ditampilkan pada Tabel **??**.

Table 3. Tabel kebenaran disjungsi.

A	B	$A \vee B$
B	B	B
B	S	B
S	B	B
S	S	S

Disjungsi di tabel ini adalah disjungsi inklusif, artinya kedua-duanya boleh benar sekaligus. Ini berbeda dari disjungsi eksklusif (XOR) yang hanya benar jika tepat satu yang benar. Di perangkat lunak, disjungsi inklusif diwakili operator `||` atau `or`. Contoh: syarat diskon adalah pelanggan baru atau member premium, dan pelanggan yang baru sekaligus premium juga berhak mendapat diskon.

3.5 Implikasi: $A \rightarrow B$

Implikasi bernilai salah hanya jika A benar tetapi B salah. Semua kombinasi lain bernilai benar, termasuk kombinasi ketika A salah. Tabel kebenarannya ditampilkan pada Tabel **??**.

Table 4. Tabel kebenaran implikasi.

A	B	$A \to B$
B	B	B
B	S	S
S	B	B
S	S	B

Implikasi adalah operator yang paling sering membingungkan. Alasannya, implikasi tidak berbicara tentang sebab-akibat, melainkan tentang janji bersyarat. Jika saya berkata "jika hujan, maka jalan basah", janji itu dilanggar hanya jika hujan turun tetapi jalan tidak basah. Jika hujan tidak turun, janji itu tidak dilanggar, apa pun kondisi jalannya. Logika menyebut keadaan ini *vacuously true*, benar karena syaratnya tidak dipenuhi. Perhatikan Gambar **??**.

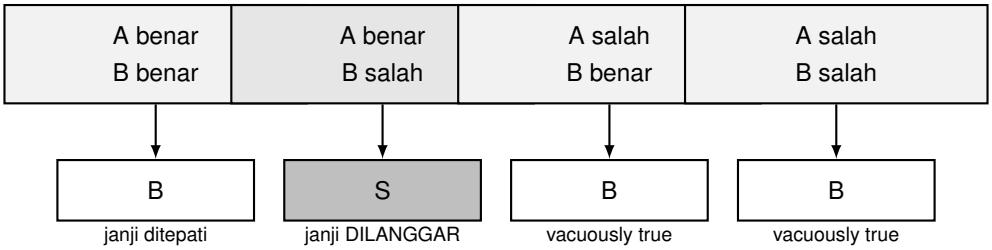


Figure 2. Empat skenario implikasi $A \to B$: hanya skenario kedua yang bernilai salah.

Cara membaca Gambar **??**: gambar ini menampilkan empat kotak berjajar di atas, masing-masing berisi kombinasi nilai A dan B. Kotak pertama berisi "A benar, B benar". Kotak kedua berisi "A benar, B salah". Kotak ketiga berisi "A salah, B benar". Kotak keempat berisi "A salah, B salah". Di bawah setiap kotak ada panah yang menunjuk ke kotak hasil berisi B atau S. Hanya kotak kedua yang hasilnya S dan diberi latar abu-abu gelap. Tiga kotak lain hasilnya B. Di bawah setiap kotak hasil ada keterangan: "janji ditepati" untuk kotak pertama, "janji DILANGGAR" untuk kotak kedua, dan "vacuously true" untuk kotak ketiga dan keempat. Inti pesan gambar ini adalah implikasi hanya salah ketika syarat terpenuhi tetapi akibat tidak terjadi.

Di perangkat lunak, implikasi muncul sebagai syarat validasi. Contoh: "jika form dikirim, maka semua field wajib terisi". Ini diterjemahkan menjadi "jika bukan form dikirim, atau semua field terisi", karena $A \to B$ setara dengan $\neg A \lor B$. Kesetaraan ini dibahas pada Bagian **??**.

3.6 Biimplikasi: $A \leftrightarrow B$

Biimplikasi bernilai benar ketika A dan B bernilai sama, keduanya benar atau keduanya salah. Tabel kebenarannya ditampilkan pada Tabel **??**.

Table 5. Tabel kebenaran biimplikasi.

A	B	$A \leftrightarrow B$
B	B	B
B	S	S
S	B	S
S	S	B

Di perangkat lunak, biimplikasi muncul sebagai perbandingan kesamaan. Contoh: "toggle lampu menyala jika dan hanya jika sakelar dalam posisi ON". Ekspresinya adalah `lampuMenyala == sakelarOn`. Operator perbandingan `==` pada boolean berperilaku seperti biimplikasi.

3.7 Ringkasan Lima Operator

Gambar **??** menyandingkan kelima tabel kebenaran dalam satu pandangan.

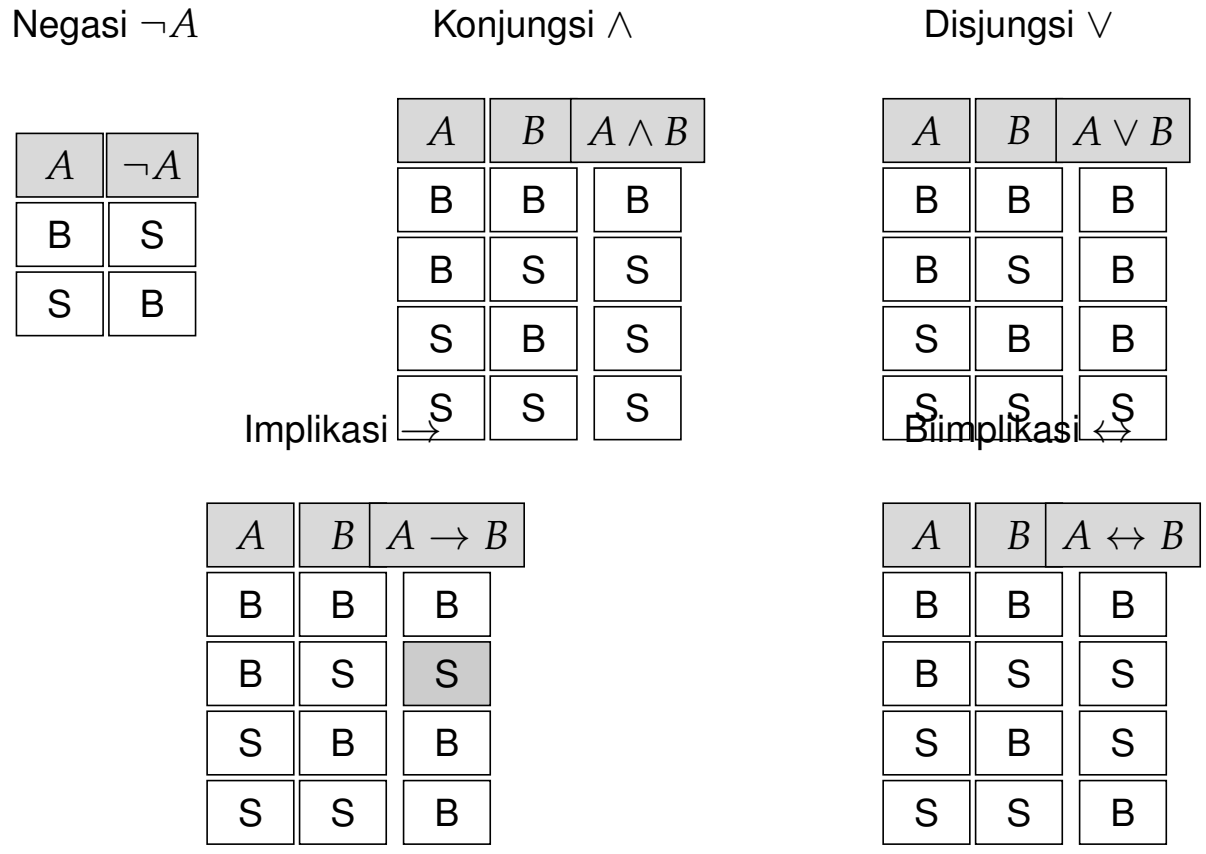


Figure 3. Kelima tabel kebenaran operator dasar dalam satu pandangan.

Cara membaca Gambar **??**: gambar ini menampilkan lima tabel kecil yang tersusun dua baris. Baris atas dari kiri ke kanan berisi tabel negasi dengan dua baris data, tabel konjungsi dengan empat baris data, dan tabel disjungsi dengan empat baris data. Baris bawah berisi tabel implikasi dan tabel biimplikasi, masing-masing dengan empat baris data. Header kolom setiap tabel diberi latar abu-abu muda. Pada tabel implikasi, sel yang bernilai S di baris kedua (ketika

A benar tetapi B salah) diberi latar abu-abu lebih gelap untuk menandakan bahwa hanya kombinasi itu yang salah. Kolom hasil setiap tabel diletakkan di kolom paling kanan. Inti pesan gambar ini adalah setiap operator punya pola kebenaran yang berbeda, dan pola itu bisa dibaca dari kolom hasil.

3.8 Ringkasan

Tabel **??** merangkum kelima operator dengan padanan di perangkat lunak.

Table 6. Ringkasan lima operator logika dan padanannya di perangkat lunak.

Nama	Simbol	Kapan benar	Padanan kode
Negasi	$\neg A$	ketika A salah	`!A`
Konjungsi	$A \wedge B$	keduanya benar	`A && B`
Disjungsi	$A \vee B$	salah satu atau keduanya benar	`A || B`
Implikasi	$A \rightarrow B$	kecuali A benar dan B salah	`!A || B`
Biimplikasi	$A \leftrightarrow B$	keduanya sama	`A == B`

4 Membangun Tabel Kebenaran Majemuk

4.1 Analogi: Menyusun Jadwal Latihan

Bayangkan seorang pelatih yang ingin menyusun jadwal latihan untuk tiga atlet. Setiap atlet bisa hadir atau tidak hadir, sehingga ada delapan kemungkinan kombinasi kehadiran. Untuk memastikan tidak ada kombinasi yang terlewat, pelatih menulis semua kemungkinan secara sistematis, dari yang paling banyak hadir sampai yang paling sedikit.

Tabel kebenaran dibangun dengan cara yang sama. Untuk n variabel, ada 2^n kombinasi. Kombinasi disusun secara sistematis agar tidak ada yang terlewat dan tidak ada yang terulang.

4.2 Aturan Dasar

Untuk formula dengan n variabel proposisi, tabel kebenaran memiliki 2^n baris data. Tabel juga memiliki $n + 1$ kolom minimal, yaitu n kolom variabel dan satu kolom hasil akhir. Jika ada sub-ekspresi di dalam formula, kolom tambahan diperlukan untuk setiap sub-ekspresi [**?**].

Ada tiga langkah menyusun tabel kebenaran. Langkah pertama, daftar semua kombinasi nilai variabel. Cara termudah adalah menganggap variabel pertama berganti setiap 2^{n-1} baris, variabel kedua berganti setiap 2^{n-2} baris, dan seterusnya. Langkah kedua, hitung sub-ekspresi dari dalam ke luar sesuai urutan operator. Langkah ketiga, isi kolom hasil akhir berdasarkan sub-ekspresi terakhir.

4.3 Contoh: $W = (\neg A \vee B) \wedge (\neg B \vee A)$

Formula ini memiliki dua variabel, sehingga ada $2^2 = 4$ baris. Sub-ekspresi yang perlu dihitung ada empat: $\neg A$, $\neg A \vee B$, $\neg B$, dan $\neg B \vee A$. Setelah itu, kolom hasil akhir dihitung dari konjungsi kedua sub-ekspresi tengah.

Baris pertama, $A = B$ dan $B = B$. $\neg A = S$, $\neg A \vee B = B$. $\neg B = S$, $\neg B \vee A = B$. Konjungsi $B \wedge B = B$.

Baris kedua, $A = B$ dan $B = S$. $\neg A = S$, $\neg A \vee B = S$. $\neg B = B$, $\neg B \vee A = B$. Konjungsi $S \wedge B = S$.

Baris ketiga, $A = S$ dan $B = B$. $\neg A = B$, $\neg A \vee B = B$. $\neg B = S$, $\neg B \vee A = S$. Konjungsi $B \wedge S = S$.

Baris keempat, $A = S$ dan $B = S$. $\neg A = B$, $\neg A \vee B = B$. $\neg B = B$, $\neg B \vee A = B$. Konjungsi $B \wedge B = B$.

Hasil akhir: B, S, S, B untuk empat baris. Perhatikan Gambar **??**.

Langkah 1. Hitung n variabel, tentukan 2^n baris

↓

Langkah 2. Daftar semua kombinasi secara sistematis

↓

Langkah 3. Hitung sub-ekspresi terdalam lebih dulu

↓

Langkah 4. Naik ke sub-ekspresi luar

↓

Langkah 5. Isi kolom hasil akhir

↓

Langkah 6. Klasifikasi: tautologi, kontradiksi, kontingensi

Figure 4. Alur enam langkah membangun tabel kebenaran majemuk.

Cara membaca Gambar **??**: gambar ini menampilkan enam kotak persegi panjang tersusun dari atas ke bawah. Setiap kotak berisi satu langkah, diawali dengan angka. Kotak pertama "Hitung n variabel, tentukan 2^n*baris*".*Kotakkedua*"*Daftarsemuakombinasisecarasistematis*".*Kotakketiga*"*Hitungsub* − *ekspresiterdalamlebihdulu*".*Kotakkeempat*"*Naikkesub* − *ekspresiluar*".*Kotakkelima*"*Isikolomhasilakhir*".*Kotakke tautologi*, *kontradiksi*, *kontingensi*".*Setiapkotakdihubungkankekotakberikutnyaolehpanahpendekkebawah*.*Latark abusangatmuda*.*Intipesangambariniadalahpenyusunantabelkebenaranberjalanberurutandaripersiapanbaris*, *per ekspresi*, *sampaiklasifikasihasil*.

4.4 Cara Berpikir: Urutan Kombinasi

Saat mengisi kolom variabel, jangan mengisi acak. Mulai dari kombinasi paling atas dengan semua variabel bernilai B, lalu turunkan satu per satu seperti menghitung biner. Variabel paling kanan berganti setiap baris, variabel kedua dari kanan berganti setiap dua baris, dan seterusnya.

Dengan pola ini, 2^n baris terisi tanpa ada yang terlewat.

Jika langkah ini dilewati dan baris diisi acak, ada risiko satu kombinasi tertulis dua kali, sedangkan kombinasi lain tidak pernah muncul. Tabel kebenaran yang tidak lengkap tidak bisa dipakai untuk membuktikan apa pun.

4.5 Kalau Salah: Tabel Tidak Lengkap

Tabel kebenaran yang tidak memuat semua 2^n kombinasi adalah tabel yang tidak berguna. Jika aturan akses punya empat variabel, ada 16 kombinasi. Tabel dengan 12 baris hanya memeriksa 75 persen kasus. Bug yang muncul pada 4 kombinasi yang hilang akan lolos dari pengujian.

Pemrogram yang mengandalkan pengujian manual biasanya hanya menguji 2 atau 3 kombinasi yang terpikir. Tabel kebenaran memaksa semua kombinasi diperiksa, sehingga bug yang tersembunyi di kombinasi jarang ikut ditemukan [**?**].

5 Tautologi, Kontradiksi, dan Kontingensi

5.1 Analogi: Tiga Jenis Janji

Bayangkan tiga jenis janji. Janji pertama selalu ditepati, apa pun kondisinya. Janji kedua selalu dilanggar, apa pun kondisinya. Janji ketiga kadang ditepati, kadang dilanggar, tergantung situasi.

Ketiga jenis janji itu adalah tautologi, kontradiksi, dan kontingensi.

5.2 Definisi Formal

Tautologi adalah ekspresi logika yang selalu bernilai benar untuk semua kombinasi nilai variabelnya. Kontradiksi adalah ekspresi yang selalu bernilai salah. Kontingensi adalah ekspresi yang kadang benar, kadang salah, tergantung nilai variabelnya [**?**].

Contoh tautologi: $A \lor \neg A$. Ini disebut *excluded middle*, karena setiap proposisi pasti benar atau salah, tidak ada jalan tengah.

Contoh kontradiksi: $A \land \neg A$. Ini disebut *contradiction*, karena sebuah proposisi tidak mungkin benar sekaligus salah.

Contoh kontingensi: $A \lor B$. Hasilnya benar pada tiga kombinasi, salah pada satu kombinasi. Perhatikan Gambar **??**.

Tautologi $A \vee \neg A$

A	$\neg A$	Hasil
B	S	B
S	B	B

Kontradiksi $A \wedge \neg A$

A	$\neg A$	Hasil
B	S	S
S	B	S

Kontingensi $A \vee B$

A	B	Hasil
B	B	B
B	S	B
S	B	B
S	S	S

Figure 5. Tiga contoh klasifikasi ekspresi: tautologi, kontradiksi, dan kontingensi.

Cara membaca Gambar **??**: gambar ini menampilkan tiga tabel kebenaran kecil berjajar dari kiri ke kanan. Tabel kiri berjudul "Tautologi A ¬A" dengan kolom hasil seluruhnya B dan diberi latar abu-abu. Tabel tengah berjudul "Kontradiksi A ¬A" dengan kolom hasil seluruhnya S dan diberi latar abu-abu. Tabel kanan berjudul "Kontingensi A B" dengan kolom hasil B, B, B, S, hanya satu baris yang S dan baris itu diberi latar abu-abu untuk menandakan baris pembeda. Inti pesan gambar ini adalah klasifikasi ditentukan oleh seluruh isi kolom hasil, bukan oleh satu baris saja.

5.3 Mengapa Klasifikasi Penting

Di perangkat lunak, tautologi adalah ekspresi yang tidak perlu ditulis karena selalu benar. Jika seorang pemrogram menulis `if (x > 0 || x <= 0)`, kondisi itu selalu benar, sehingga blok `else` tidak pernah dieksekusi. Menyadari bahwa itu tautologi membuat pemrogram menghapus kondisi yang tidak berguna.

Kontradiksi adalah ekspresi yang tidak akan pernah benar. Jika seorang pemrogram menulis `if (x > 0 && x < 0)`, blok itu tidak akan pernah dieksekusi. Menyadari itu kontradiksi membuat pemrogram menyadari ada bug.

Kontingensi adalah ekspresi normal yang bergantung pada nilai. Ekspresi inilah yang biasanya dipakai.

5.4 Cara Berpikir: Mengklasifikasi Ekspresi

Untuk mengklasifikasi ekspresi, bangun tabel kebenaran sampai selesai, lalu periksa kolom hasil. Jika semua baris bernilai B, ekspresi itu tautologi. Jika semua baris bernilai S, ekspresi itu kontradiksi. Jika campuran, ekspresi itu kontingensi.

Jangan mengklasifikasi dari beberapa baris pertama. Satu baris yang berbeda di baris terakhir sudah cukup untuk mengubah klasifikasi dari tautologi menjadi kontingensi.

6 Hukum-Hukum Logika

6.1 Analogi: Alat Tukang untuk Membongkar Ekspresi

Bayangkan seorang tukang kayu dengan kotak perkakas berisi berbagai alat. Setiap alat punya kegunaan khusus. Palu untuk memukul, obeng untuk memutar sekrup, gergaji untuk memotong. Tukang kayu yang mengenal alatnya dapat memilih alat yang tepat tanpa harus memaksakan satu alat untuk semua pekerjaan.

Hukum-hukum logika adalah kotak perkakas itu. Setiap hukum mengubah ekspresi dengan cara yang berbeda. Pemrogram yang mengenal hukumnya dapat menyederhanakan ekspresi tanpa mengubah perilaku program.

6.2 Sembilan Hukum Dasar

Komutatif: $A \wedge B \equiv B \wedge A$ dan $A \vee B \equiv B \vee A$. Urutan operand tidak berpengaruh pada konjungsi dan disjungsi.

Asosiatif: $(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$ dan $(A \vee B) \vee C \equiv A \vee (B \vee C)$. Pengelompokan tiga operand tidak berpengaruh.

Distributif: $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$ dan $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$. Konjungsi menyebar ke dalam disjungsi, dan sebaliknya.

Identitas: $A \wedge B \equiv A$ jika B adalah Benar, dan $A \vee S \equiv A$ jika S adalah Salah. Benar adalah identitas untuk konjungsi, Salah adalah identitas untuk disjungsi.

Idempoten: $A \wedge A \equiv A$ dan $A \vee A \equiv A$. Mengulang operand yang sama tidak mengubah hasil.

Dominasi: $A \vee B \equiv B$ jika B adalah Benar, dan $A \wedge S \equiv S$ jika S adalah Salah. Benar mendominasi disjungsi, Salah mendominasi konjungsi.

Negasi Ganda: $\neg(\neg A) \equiv A$. Membalik dua kali mengembalikan nilai asli.

Excluded Middle: $A \vee \neg A \equiv$ Benar dan $A \wedge \neg A \equiv$ Salah. Sebuah proposisi pasti benar atau salah, tidak keduanya.

De Morgan: $\neg(A \wedge B) \equiv \neg A \vee \neg B$ dan $\neg(A \vee B) \equiv \neg A \wedge \neg B$. Negasi masuk ke dalam dengan membalik operator. Perhatikan Gambar **??**.

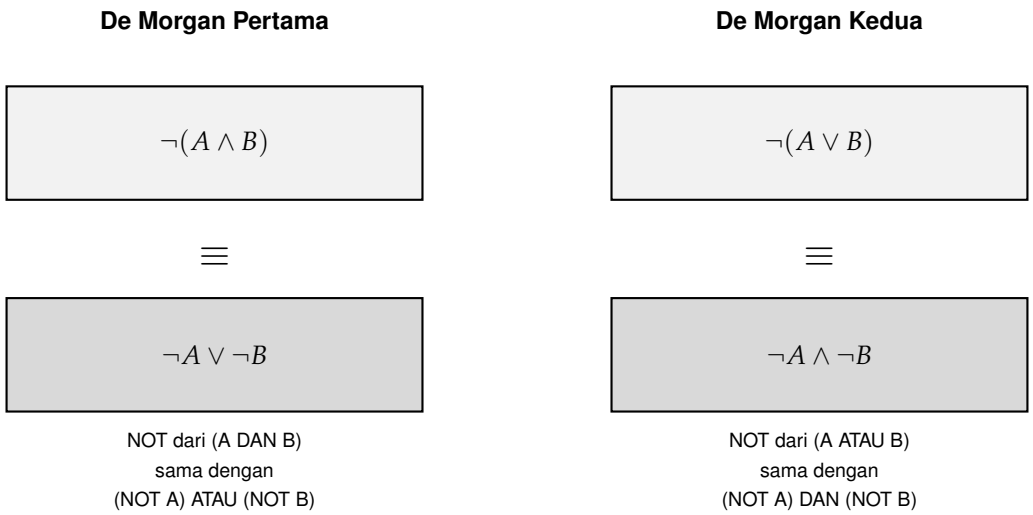


Figure 6. Dua bentuk De Morgan: negasi masuk ke dalam dan operator dibalik.

Cara membaca Gambar **??**: gambar ini menampilkan dua kolom berjajar. Kolom kiri berjudul "De Morgan Pertama" dengan tiga kotak bertumpuk. Kotak atas berisi $\neg(A \wedge B)$, kotak tengah berisi tanda $\equiv$, kotak bawah berisi $\neg A \vee \neg B$. Di bawahnya ada keterangan "NOT dari (A DAN B) sama dengan (NOT A) ATAU (NOT B)". Kolom kanan berjudul "De Morgan Kedua" dengan pola serupa: $\neg(A \vee B)$ di atas, tanda $\equiv$ di tengah, $\neg A \wedge \neg B$ di bawah, dan keterangan "NOT dari (A ATAU B) sama dengan (NOT A) DAN (NOT B)". Kotak bawah kedua kolom diberi latar abu-abu lebih gelap untuk menandakan bentuk hasil. Inti pesan gambar ini adalah De Morgan menukar peran DAN dan ATAU sambil memindahkan negasi ke setiap operand.

6.3 Ringkasan

Tabel **??** merangkum sembilan hukum logika.

Table 7. Ringkasan hukum-hukum logika proposisi.

Nama hukum	Bentuk
Komutatif	$A \wedge B \equiv B \wedge A$; $A \vee B \equiv B \vee A$
Asosiatif	$(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$
Distributif	$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
Identitas	$A \wedge \text{Benar} \equiv A$; $A \vee \text{Salah} \equiv A$
Idempoten	$A \wedge A \equiv A$; $A \vee A \equiv A$
Dominasi	$A \vee \text{Benar} \equiv \text{Benar}$; $A \wedge \text{Salah} \equiv \text{Salah}$
Negasi Ganda	$\neg(\neg A) \equiv A$
Excluded Middle	$A \vee \neg A \equiv \text{Benar}$; $A \wedge \neg A \equiv \text{Salah}$
De Morgan	$\neg(A \wedge B) \equiv \neg A \vee \neg B$; $\neg(A \vee B) \equiv \neg A \wedge \neg B$

6.4 Mengapa Hukum Logika Penting

Hukum logika dipakai untuk menyederhanakan ekspresi tanpa harus membangun tabel kebenaran. Jika ekspresi awal punya 4 variabel, tabel kebenarannya 16 baris. Jika ekspresi bisa disederhanakan menjadi 2 variabel, tabel kebenarannya hanya 4 baris. Pengujian menjadi lebih cepat dan lebih mudah dibaca.

Selain itu, hukum logika dipakai untuk membuktikan bahwa dua ekspresi ekivalen. Bukti dengan tabel kebenaran memerlukan 16 baris, sedangkan bukti dengan hukum logika hanya memerlukan beberapa langkah.

7 Ekivalensi Logis dan De Morgan

7.1 Analogi: Dua Resep yang Hasilnya Sama

Bayangkan dua resep kue. Resep pertama memakai mentega dan gula. Resep kedua memakai gula dan mentega. Kedua resep itu berbeda urutan bahannya, tetapi menghasilkan kue yang sama.

Dua ekspresi logika disebut ekivalen jika keduanya selalu bernilai sama untuk setiap kombinasi nilai variabelnya. Notasinya $W_1 \equiv W_2$.

7.2 Cara Membuktikan Ekivalensi

Ada dua cara membuktikan ekivalensi. Cara pertama dengan tabel kebenaran: bangun tabel untuk kedua ekspresi, lalu bandingkan kolom hasilnya. Jika setiap baris bernilai sama, kedua ekspresi ekivalen. Cara kedua dengan hukum logika: ubah ekspresi pertama langkah demi langkah sampai bentuknya sama dengan ekspresi kedua.

Contoh klasik: buktikan $A \to B \equiv \neg A \lor B$. Bukti dengan tabel kebenaran ditampilkan pada Tabel **??**.

Table 8. Bukti ekivalensi $A \to B \equiv \neg A \lor B$ dengan tabel kebenaran.

A	B	$A \to B$	$\neg A$	$\neg A \lor B$
B	B	B	S	B
B	S	S	S	S
S	B	B	B	B
S	S	B	B	B

Kolom ketiga dan kolom kelima identik di setiap baris, sehingga kedua ekspresi ekivalen. Ekivalensi ini dipakai di perangkat lunak untuk menuliskan implikasi dalam bentuk yang lebih mudah dievaluasi oleh komputer.

7.3 De Morgan di Perangkat Lunak

De Morgan sering muncul saat menyederhanakan kondisi negasi. Contoh: `!(isLoggedIn && isAdmin)` setara dengan `!isLoggedIn || !isAdmin`. Keduanya bernilai benar ketika pengguna bukan admin atau belum login.

De Morgan juga dipakai untuk mengubah kondisi menjadi bentuk yang lebih mudah dibaca. Contoh: "pengguna tidak berhak mengakses" bisa ditulis sebagai $\neg(L \wedge A)$ atau sebagai $\neg L \vee \neg A$. Kedua bentuk itu sama secara logika, tetapi bentuk kedua lebih jelas menjelaskan dua alasan penolakan: belum login, atau bukan admin.

7.4 Cara Berpikir: Memilih Cara Bukti

Jika ekspresi hanya punya 2 variabel, tabel kebenaran dengan 4 baris lebih cepat. Jika ekspresi punya 3 variabel, tabel dengan 8 baris masih wajar. Jika ekspresi punya 4 variabel atau lebih, tabel dengan 16 baris atau lebih mulai melelahkan. Pada kasus itu, bukti dengan hukum logika lebih efisien, walaupun memerlukan pemahaman hukum.

8 Menyederhanakan Ekspresi

8.1 Analogi: Membongkar Kabel yang Kusut

Bayangkan sebuah kotak kabel yang kusut. Ada empat kabel yang saling melilit. Setelah diurai dengan teliti, ternyata keempat kabel itu hanyalah satu kabel yang dilipat empat kali. Kotak yang tampak rumit ternyata hanya berisi satu kabel sederhana.

Penyederhanaan ekspresi logika bekerja seperti mengurai kabel itu. Ekspresi yang tampak rumit diurai dengan hukum logika, dan ternyata hanya setara dengan satu kondisi sederhana.

8.2 Contoh Lengkap

Sederhanakan $\neg(\neg A \vee B) \vee (A \wedge B)$. Setiap langkah dijelaskan dengan nama hukum yang dipakai.

Ekspresi awal: $\neg(\neg A \vee B) \vee (A \wedge B)$.

Langkah 1, De Morgan pada $\neg(\neg A \vee B)$: $\neg(\neg A \vee B) \equiv \neg(\neg A) \wedge \neg B$. Bentuk sekarang: $(\neg(\neg A) \wedge \neg B) \vee (A \wedge B)$.

Langkah 2, Negasi Ganda pada $\neg(\neg A)$: $\neg(\neg A) \equiv A$. Bentuk sekarang: $(A \wedge \neg B) \vee (A \wedge B)$.

Langkah 3, Distributif: $A \wedge \neg B$ dan $A \wedge B$ punya faktor sama A, sehingga bisa difaktorkan menjadi $A \wedge (\neg B \vee B)$.

Langkah 4, Excluded Middle pada $\neg B \vee B$: $\neg B \vee B \equiv$ Benar. Bentuk sekarang: $A \wedge$ Benar.

Langkah 5, Identitas pada $A \wedge \text{Benar}$: $A \wedge \text{Benar} \equiv A$.

Hasil akhir: A. Ekspresi yang tampak rumit dengan tiga operator ternyata hanya setara dengan satu variabel. Perhatikan Gambar **??**.

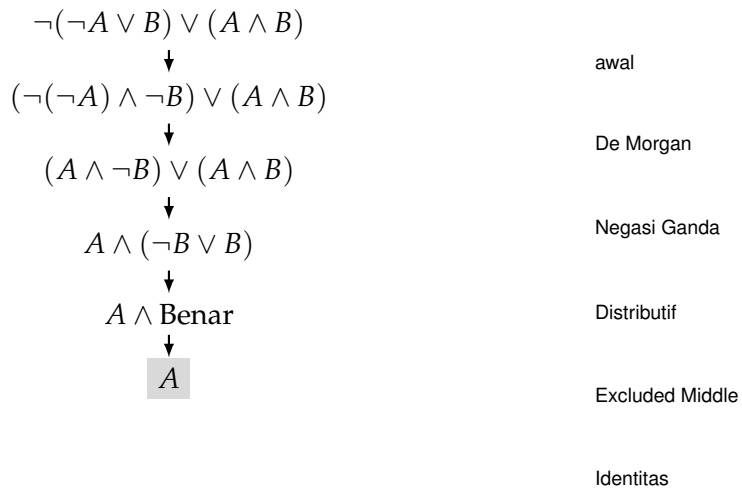


Figure 7. Alur penyederhanaan ekspresi dengan nama hukum di setiap langkah.

Cara membaca Gambar **??**: gambar ini menampilkan enam kotak persegi panjang tersusun dari atas ke bawah. Setiap kotak berisi satu bentuk ekspresi. Kotak pertama ekspresi awal $\neg(\neg A \vee B) \vee (A \wedge B)$. Kotak kedua hasil De Morgan. Kotak ketiga hasil Negasi Ganda. Kotak keempat hasil Distributif. Kotak kelima hasil Excluded Middle. Kotak keenam hasil Identitas, yaitu A, dengan latar abu-abu lebih gelap untuk menandakan hasil akhir. Di sebelah kanan setiap kotak ada label nama hukum yang dipakai. Panah menghubungkan kotak dari atas ke bawah. Inti pesan gambar ini adalah setiap langkah penyederhanaan dijelaskan oleh satu nama hukum, dan hasil akhirnya jauh lebih sederhana dari ekspresi awal.

8.3 Mengapa Penyederhanaan Penting

Kode warisan (legacy code) sering berisi kondisi boolean yang berlebihan. Ekspresi yang sama ditulis berkali-kali dengan bentuk berbeda. Penyederhanaan dengan hukum logika membuat kode lebih pendek, lebih mudah dibaca, dan lebih mudah diuji, tanpa mengubah perilaku program.

Selain itu, penyederhanaan mengurangi jumlah kasus uji. Ekspresi dengan 4 variabel memerlukan 16 kasus uji. Ekspresi yang sudah disederhanakan menjadi 1 variabel hanya memerlukan 2 kasus uji. Penghematan waktu pengujian ini sangat berarti pada sistem besar [**?**].

8.4 Kalau Salah: Menyederhanakan dengan Menghapus Operator

Kesalahan paling sering adalah menyederhanakan dengan menghapus operator yang tampak tidak perlu. Contoh: mengira $\neg(A \wedge B)$ sama dengan $\neg A \wedge \neg B$. Bentuk kedua itu salah, karena De Morgan mengubah $\wedge$ menjadi $\vee$.

Perbedaan ini bukan sepele. Jika $A = B$ dan $B = S$, maka $\neg(A \wedge B) = \neg(B \wedge S) = \neg S = B$, sedangkan $\neg A \wedge \neg B = S \wedge B = S$. Kedua hasil berbeda. Kode yang memakai bentuk salah akan menolak pengguna yang seharusnya diterima, atau menerima pengguna yang seharusnya ditolak.

9 Studi Kasus 1: Validasi Akses Dashboard Admin

9.1 Langkah 1: Definisikan Proposisi

Sistem Informasi Akademik memiliki halaman dashboard admin yang hanya boleh dibuka oleh pengguna tertentu. Ada empat proposisi yang terlibat. L berarti "login pengguna valid". A berarti "pengguna berperan sebagai admin". O berarti "kode OTP tervalidasi". B berarti "akun sedang diblokir".

Keempat proposisi itu bukan variabel bebas. Setiap pengguna punya nilai pasti untuk keempatnya. Login bisa valid atau tidak valid. Peran bisa admin atau bukan admin. OTP bisa tervalidasi atau tidak. Akun bisa diblokir atau tidak.

9.2 Langkah 2: Rumuskan Aturan

Aturan dalam bahasa natural: "Akses diizinkan jika pengguna login valid, berperan admin, OTP tervalidasi, dan akun tidak sedang diblokir."

Aturan ini punya empat syarat yang harus dipenuhi semua. Kata "dan" menandakan konjungsi. Syarat keempat adalah "tidak sedang diblokir", yang merupakan negasi dari B.

9.3 Langkah 3: Terjemahkan ke Ekspresi Logika

Ekspresi logikanya adalah Akses $= (L \wedge A \wedge O) \wedge \neg B$. Tanda kurung mengelompokkan tiga syarat pertama, lalu hasilnya dikonjungsi dengan negasi syarat keempat. Perhatikan Gambar **??**.

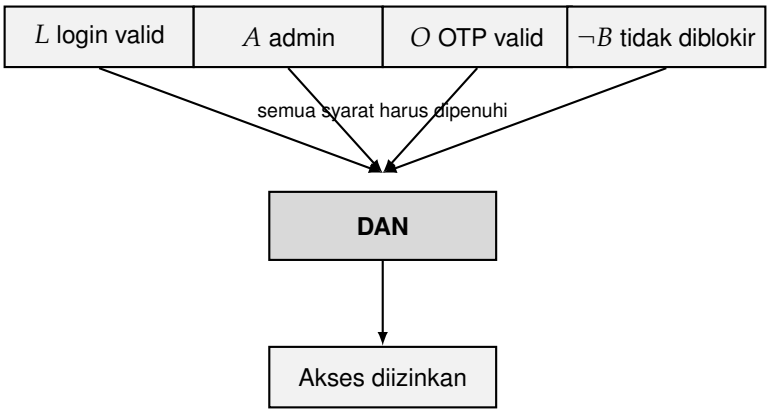


Figure 8. Gerbang logika untuk aturan akses dashboard admin.

Cara membaca Gambar **??**: gambar ini menampilkan empat kotak kondisi di bagian atas, berjajar dari kiri ke kanan: L login valid, A admin, O OTP valid, dan $\neg B$ tidak diblokir. Keempat panah dari kotak kondisi menuju ke satu kotak tengah bertuliskan "DAN" dengan latar abu-abu gelap. Di bawah kotak "DAN" ada panah menuju kotak bawah bertuliskan "Akses diizinkan". Di atas kotak "DAN" ada keterangan "semua syarat harus dipenuhi". Inti pesan gambar ini adalah aturan akses adalah gerbang DAN dengan empat masukan, dan pintu terbuka hanya jika keempatnya bernilai benar.

9.4 Langkah 4: Susun Tabel Kebenaran 16 Baris

Ada empat variabel, sehingga ada $2^4 = 16$ baris. Tabel lengkapnya ditampilkan pada Tabel **??**.

Table 9. Tabel kebenaran lengkap validasi akses dashboard admin.

#	L	A	O	B	Akses
1	B	B	B	B	S
2	B	B	B	S	B
3	B	B	S	B	S
4	B	B	S	S	S
5	B	S	B	B	S
6	B	S	B	S	S
7	B	S	S	B	S
8	B	S	S	S	S
9	S	B	B	B	S
10	S	B	B	S	S
11	S	B	S	B	S
12	S	B	S	S	S
13	S	S	B	B	S
14	S	S	B	S	S
15	S	S	S	B	S
16	S	S	S	S	S

9.5 Langkah 5: Identifikasi Baris yang Diizinkan

Dari enam belas baris, hanya baris kedua yang bernilai B pada kolom Akses. Baris kedua adalah kombinasi $L = B$, $A = B$, $O = B$, dan $B = S$. Artinya, pengguna diizinkan masuk hanya ketika login valid, berperan admin, OTP tervalidasi, dan akun tidak diblokir. Semua kombinasi lain menolak akses.

Baris pertama menarik untuk diperhatikan. Pada baris pertama, semua syarat positif terpenuhi, yaitu $L = B$, $A = B$, dan $O = B$. Namun $B = B$ juga, artinya akun sedang diblokir. Akses pada baris pertama tetap S. Ini menunjukkan bahwa akun yang diblokir tetap ditolak, walaupun semua syarat lain terpenuhi. Tanpa tanda negasi pada B, baris pertama akan bernilai B, dan akun yang diblokir bisa masuk.

9.6 Langkah 6: Tulis Kode Program

Kode program yang mengimplementasikan aturan ini ditulis dengan operator yang sesuai. Konjungsi menjadi `&&`, negasi menjadi `!`.

```
if (isLoggedIn && isAdmin && otpValid && !isBlocked) {
    showAdminDashboard();
}
```

Kode itu persis mengikuti ekspresi $(L \wedge A \wedge O) \wedge \neg B$. Tidak ada operator yang ditambah, tidak ada yang dihilangkan.

9.7 Langkah 7: Verifikasi dengan Negasi

Cara lain memeriksa kebenaran aturan adalah memeriksa negasinya. Pengguna yang ditolak adalah negasi dari aturan akses, yaitu $\neg((L \wedge A \wedge O) \wedge \neg B)$. Dengan De Morgan, ekspresi itu menjadi $\neg L \vee \neg A \vee \neg O \vee B$. Artinya, pengguna ditolak jika tidak login, atau bukan admin, atau OTP tidak valid, atau akun diblokir.

Kedua bentuk itu ekivalen. Jika pemrogram menulis aturan penolakan, ia bisa memakai bentuk negasi. Jika ia menulis aturan penerimaan, ia bisa memakai bentuk asli. Keduanya memberi hasil yang sama.

9.8 Langkah 8: Kesimpulan Kasus

Aturan akses dashboard admin adalah konjungsi dari empat syarat, dengan satu syarat dinegasikan. Tabel kebenaran membuktikan aturan itu hanya mengizinkan satu dari enam belas kombinasi, dan kombinasi yang diizinkan adalah kombinasi yang masuk akal. Tanpa tabel kebenaran, pemrogram hanya akan menguji beberapa kombinasi, dan kemungkinan besar melewatkan kombinasi yang diblokir tetapi semua syarat lain terpenuhi. Alur lengkap kasus ditunjukkan pada Gambar **??**.

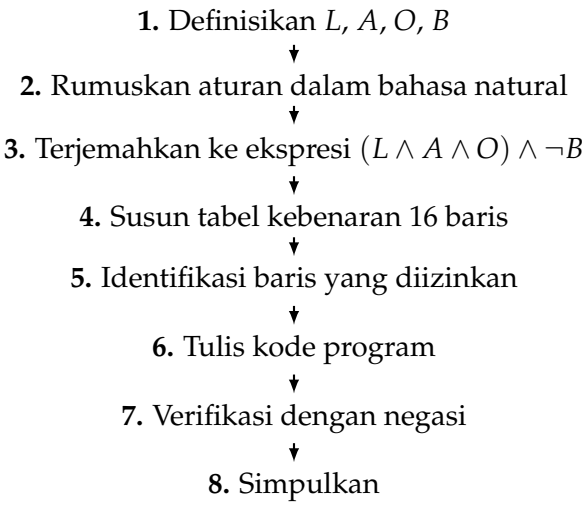


Figure 9. Alur delapan langkah studi kasus validasi akses dashboard admin.

Cara membaca Gambar **??**: gambar ini menampilkan delapan kotak persegi panjang tersusun dari atas ke bawah. Setiap kotak berisi satu langkah, diawali angka tebal. Kotak pertama "Definisikan L, A, O, B". Kotak kedua "Rumuskan aturan dalam bahasa natural". Kotak ketiga "Terjemahkan ke ekspresi (L A O) ¬B". Kotak keempat "Susun tabel kebenaran 16 baris". Kotak kelima "Identifikasi baris yang diizinkan". Kotak keenam "Tulis kode program". Kotak ketujuh "Verifikasi dengan negasi". Kotak kedelapan "Simpulkan". Panah pendek ke bawah menghubungkan setiap kotak ke kotak berikutnya. Latar kotak abu-abu sangat muda. Inti pesan gambar ini adalah perancangan aturan akses berjalan berurutan dari definisi proposisi sampai verifikasi.

10 Studi Kasus 2: Validasi Peminjaman Buku

10.1 Konteks

Perpustakaan digital memiliki formulir peminjaman buku. Peminjaman diizinkan jika empat syarat terpenuhi. K berarti "kartu anggota aktif". T berarti "tidak ada tunggakan denda". S berarti "buku berstatus tersedia". M berarti "jumlah pinjaman belum melebihi batas maksimum".

10.2 Rumusan Ekspresi

Aturan peminjaman adalah konjungsi keempat syarat: Pinjam $= K \land T \land S \land M$. Tidak ada syarat yang dinegasikan, karena keempatnya dinyatakan dalam bentuk positif.

10.3 Tabel Kebenaran 16 Baris

Dengan empat variabel, ada 16 baris. Hasilnya ditampilkan pada Tabel **??**.

Table 10. Tabel kebenaran validasi peminjaman buku.

#	K	T	S	M	Pinjam
1	B	B	B	B	B
2	B	B	B	S	S
3	B	B	S	B	S
4	B	B	S	S	S
5	B	S	B	B	S
6	B	S	B	S	S
7	B	S	S	B	S
8	B	S	S	S	S
9	S	B	B	B	S
10	S	B	B	S	S
11	S	B	S	B	S
12	S	B	S	S	S
13	S	S	B	B	S
14	S	S	B	S	S
15	S	S	S	B	S
16	S	S	S	S	S

Hanya baris pertama bernilai B. Artinya, peminjaman diizinkan hanya ketika keempat syarat terpenuhi. Bandingkan dengan studi kasus sebelumnya: pada dashboard admin, hanya baris kedua yang diizinkan karena ada negasi pada B. Pada peminjaman buku, semua syarat dinyatakan positif, sehingga baris pertama yang diizinkan.

10.4 Klasifikasi Ekspresi

Ekspresi $K \wedge T \wedge S \wedge M$ adalah kontingensi. Kolom hasilnya berisi satu B dan lima belas S. Ekspresi ini tidak selalu benar, tidak selalu salah, tetapi bergantung pada nilai keempat variabel.

Ekspresi ini juga bukan tautologi, karena ada baris yang hasilnya S. Bukan kontradiksi, karena ada baris yang hasilnya B.

10.5 Penyederhanaan

Ekspresi $K \wedge T \wedge S \wedge M$ sudah dalam bentuk paling sederhana. Tidak ada hukum logika yang bisa mengurangi jumlah variabel, karena keempat variabel muncul satu kali dan tidak ada pasangan yang saling meniadakan. Setiap variabel berperan independen dalam menentukan hasil.

Jika pemrogram mencoba menyederhanakan dengan menghapus salah satu variabel, perilaku program berubah. Tanpa variabel K, mahasiswa yang kartunya tidak aktif tetap bisa meminjam. Itu tidak sesuai aturan perpustakaan.

10.6 Mengapa Kasus Kedua Penting

Studi kasus kedua menunjukkan bahwa tidak semua ekspresi bisa disederhanakan. Ada ekspresi yang sudah minimal, dan menyederhanakannya lebih lanjut akan mengubah perilaku. Pemrogram yang terobsesi menyederhanakan tanpa memeriksa tabel kebenaran bisa menghapus syarat yang penting.

Kedua studi kasus membentuk pasangan yang berguna. Yang pertama memiliki negasi, sehingga tabel kebenaran menunjukkan baris kedua yang diizinkan. Yang kedua tidak memiliki negasi, sehingga tabel kebenaran menunjukkan baris pertama yang diizinkan. Keduanya menyampaikan pesan yang sama: tabel kebenaran memaksa pemrogram memeriksa semua kombinasi, bukan hanya kombinasi yang terpikir.

11 Kesalahan Umum dan Cara Menghindarinya

Tabel **??** memuat kesalahan yang paling sering muncul saat bekerja dengan logika proposisi.

Table 11. Kesalahan umum pada penerapan logika proposisi.

Kesalahan	Akibat	Perbaikan
menganggap pertanyaan atau perintah sebagai proposisi	ekspresi tidak bisa dinilai	pastikan kalimat berita dan variabel sudah diberi nilai
menulis $\wedge$ padahal maksudnya $\vee$	aturan terlalu ketat	tentukan apakah semua atau salah satu
menulis $\vee$ padahal maksudnya $\wedge$	aturan terlalu longgar	tentukan apakah semua atau salah satu
menganggap $\vee$ sebagai XOR	aturan menolak pelanggan yang memenuhi dua syarat	disjungsi inklusif, bukan eksklusif
menafsirkan implikasi sebagai sebab-akibat	bingung saat A salah	implikasi hanya janji bersyarat
lupa negasi pada syarat blokir	akun diblokir tetap bisa masuk	periksa setiap syarat negatif
De Morgan salah operator	hasil kebalikan	$\wedge$ menjadi $\vee$, $\vee$ menjadi $\wedge$
menyederhanakan tanpa cek tabel	perilaku program berubah	verifikasi dengan tabel kebenaran
menguji hanya kasus normal	bug di kombinasi jarang lolos	bangun tabel kebenaran lengkap
kurang tanda kurung di kode	presedensi operator salah	pakai kurung eksplisit

Dua kesalahan yang paling sering adalah melupakan negasi pada syarat blokir, dan menulis kode tanpa tanda kurung eksplisit. Kesalahan pertama membuat akun yang diblokir bisa masuk. Kesalahan kedua membuat `a || b && c` dievaluasi sebagai `a || (b && c)` padahal maksudnya `(a || b) && c`. Tanda kurung eksplisit mencegah kebingungan yang berasal dari presedensi operator.

12 Lembar Rujukan Cepat

Tabel **??** adalah lembar rujukan untuk ujian atau kerja.

Table 12. Lembar rujukan cepat logika proposisi.

Kebutuhan	Notasi atau aturan
proposisi	kalimat berita bernilai B atau S
negasi	$\neg A$, benar jika A salah
konjungsi	$A \wedge B$, benar jika keduanya benar
disjungsi	$A \vee B$, benar jika salah satu benar
implikasi	$A \rightarrow B$, salah hanya jika A benar dan B salah
biimplikasi	$A \leftrightarrow B$, benar jika keduanya sama
jumlah baris tabel	2^n untuk n variabel
komutatif	$A \wedge B \equiv B \wedge A$
asosiatif	$(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$
distributif	$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
identitas	$A \wedge \mathrm{B} \equiv A$; $A \vee \mathrm{S} \equiv A$
idempoten	$A \wedge A \equiv A$
dominasi	$A \vee \mathrm{B} \equiv \mathrm{B}$; $A \wedge \mathrm{S} \equiv \mathrm{S}$
negasi ganda	$\neg(\neg A) \equiv A$
excluded middle	$A \vee \neg A \equiv \mathrm{B}$; $A \wedge \neg A \equiv \mathrm{S}$
De Morgan pertama	$\neg(A \wedge B) \equiv \neg A \vee \neg B$
De Morgan kedua	$\neg(A \vee B) \equiv \neg A \wedge \neg B$
ekivalensi implikasi	$A \rightarrow B \equiv \neg A \vee B$
tautologi	semua baris B
kontradiksi	semua baris S
kontingensi	campuran B dan S

13 Kesimpulan

Modul ini menunjukkan bahwa logika proposisi adalah alat kerja untuk mengubah aturan keputusan menjadi ekspresi yang dapat diperiksa dan diuji. Tiga prinsip penutup adalah sebagai berikut.

1. Nyatakan setiap syarat sebagai proposisi yang punya nilai pasti. Tanpa proposisi, ekspresi logika tidak bisa dinilai, dan program tidak dapat diprediksi.
2. Bangun tabel kebenaran untuk semua 2^n kombinasi. Pengujian manual hanya memeriksa kombinasi yang terpikir. Tabel kebenaran memeriksa kombinasi yang tidak terpikir.
3. Sederhanakan dengan nama hukum yang jelas. Setiap langkah penyederhanaan harus menyebut nama hukum yang dipakai, sehingga hasilnya dapat diperiksa orang lain.

Dengan tiga prinsip itu, pembaca dapat menuliskan aturan keputusan program, membuktikan bahwa aturan itu benar untuk semua kombinasi kondisi, menyederhanakan ekspresi yang tampak rumit, dan merancang kode yang lebih mudah dibaca dan diuji.

Referensi

References

[1] Rosen, K. H. (2019). *Discrete Mathematics and Its Applications* (8th ed.). McGraw-Hill Education.

[2] Munir, R. (2012). *Matematika Diskrit*. Informatika.

[3] Epp, S. S. (2010). *Discrete Mathematics with Applications* (4th ed.). Brooks Cole.

[4] Lipschutz, S., & Lipson, M. (2007). *Discrete Mathematics*. McGraw-Hill.

[5] O'Regan, G. (2023). *Mathematical Foundations of Software Engineering: A Practical Guide to Essentials*. Springer.

[6] Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson.

[7] Kleppmann, M. (2017). *Designing Data-Intensive Applications*. O'Reilly Media.

[8] Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Database System Concepts* (7th ed.). McGraw-Hill Education.