МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

А. М. Сергієнко,
А. А. Молчанова,
В. О. Романкевич

КОМП'ЮТЕРНА ДИСКРЕТНА МАТЕМАТИКА

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Інженерія програмного забезпечення комп'ютерних систем»
спеціальності 121 «Інженерія програмного забезпечення» та освітньої програми: «Системне програмування та спеціалізовані комп'ютерні системи» спеціальності 123 «Комп'ютерні системи та мережі»

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2022

Рецензент *Малежик П. М.* кандидат фізико-математичних наук, доктор педагогічних наук, доцент кафедри комп'ютерної та програмної інженерії НПУ імені М.П. Драгоманова

Відповідальний редактор *Лесик Т. М.*

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 1 від 02.09.2022 р.)
за поданням Вченої ради факультету інформатики та обчислювальної техніки
(протокол № 11 від 11.07.2022 р.)

У навчальному посібнику викладено основи дискретної математики, які потрібні для формування світогляду та базових знань науковців і спеціалістів у галузі інформаційних технологій. Посібник містить основні поняття і структуру дискретної математики, розділи, які присвячені теорії множин, булевій алгебрі, абстрактним цифровим автоматам. Приділено увагу розкриттю поняття алгоритму, його зв'язку з програмуванням та проєктуванням цифрових схем. Наведено приклади програмної та апаратної реалізації об'єктів, відношень, функцій дискретної математики у програмах, дискретних схемах, комп'ютерних архітектурах. Запропоновано ієрархічний підхід до вивчення дискретної математики, починаючи з об'єктів і відношень між ними і кінчаючи алгоритмами.

Посібник призначений для здобувачів ступеня бакалавра за спеціальністю 121 «Інженерія програмного забезпечення» та спеціальністю 123 «Комп'ютерні системи та мережі», буде також корисним для студентів та спеціалістів інформаційних технологій.

Реєстр. № НП 22/23-012 Обсяг 8,4 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

Зміст

Вступ

1. Предмет дискретної математики

Викладання комп'ютерної дискретної математики в університеті має кілька аспектів:

Академічний. Дискретна математика — це точна наука, яка дає фундаментальні знання. Читання цього курсу входить у затверджений навчальний план, виконання якого студентом є обов'язковим. Мета — ознайомити студентів із базовими поняттями та методами дискретної математики.

Професійний. У значному обсязі, знання з дискретної математики — це алгоритми розв'язання більшості типових задач, основа для їх програмної чи апаратної реалізації при виконанні професійної діяльності випускників університету. Мета — сформувати навички використання методів дискретної математики.

Інтелектуальний та виховний. Вивчення знань із дискретної математики сприяє розвитку інтуїції, когнітивних навичок — суттєвого інтелектуального фактора процесу створення нових алгоритмів, матзабезпечення, винаходів.

Програмістська діяльність полягає в організації обчислювальних процесів у моделі комп'ютера за допомогою певної мови програмування. Тому виховання правильного уявлення про алгоритми, як про обчислювальні процеси, так і про моделі, у яких ці процеси відбуваються, має велике значення. Комп'ютерна дискретна математика вивчає саме об'єкти, алгоритми, моделі, які використовуються у реальних програмах.

Науки можна розділити на природничі, соціальні і точні, такі як математика.

Природничі та ***соціальні*** науки характерні тим, що в них міркування вважаються правильними, якщо результат, який одержано за

їх допомогою, співпадає з фактами, які реально спостерігаються і неправильними, якщо цей результат протирічить експериментам.

Математика ґрунтується на тому, що міркування є правильними, якщо вони є логічно бездоганними, і правила виводу, які встановлені самим математиком і які відповідають математичній логіці, додержуються на всіх етапах міркування.

Отже, математична істина залежить від коректності правил виводу і від згоди математиків, що вона одержана за правильними міркуваннями.

Математика — це наука, яка вивчає абстрактні, аксіоматичні структури за допомогою логічних правил виведення.

Дискретна математика — математика, яка вивчає математичні структури з дискретними елементами, тобто з такими, які можна чітко розрізнити один від одного. Дискретна математика — це велика галузь науки, яка має кілька розгалужень (Рис. 1).

Рис. 1. Дискретна математика та її розгалуження

У курсі комп’ютерної дискретної математики ми будемо розглядати наступні теми. Теорія множин, зокрема, відношення та функції. Булева алгебра, яка дає змогу виконувати синтез логічних комп’ютерних схем. Теорія автоматів, яка вивчає завдання алгоритмів на автоматній моделі.

2. Історія математики

Диявол грає людиною, якщо вона не мислить точно.

Р. Декарт.

Для розуміння того, навіщо дискретна математика потрібна людству, чим вона відрізняється від інших наук, яка її структура, варто коротко розглянути її історію.

Лічбою та геометричними вимірюваннями людство займалось з давніх часів. Як предмети древньої культури до нас дійшли папіруси Стародавнього Єгипту 2000-1800 рр. до н.е. — це більше 100 задач із розв’язками.

Це також глиняні таблички Стародавнього Вавилону 1800-1600 рр. до н.е. — близько 150 табличок із математичними текстами та 200 з числовими таблицями які поки знайдено і прочитано серед сотень тисяч, що зберігаються у музеях та бібліотеках. На глиняній табличці з Плімптонської колекції Колумбійського університету зображено синуси, косинуси кутів через 1 градус, які задані піфагоровими трійками.

Але це була ще не математика, тобто наука на основі логічних правил виводу. Це була більше якась природнича наука прикладного характеру, в якій знання одержувались із досвіду і використовувались у практиці.

Давньогрецька історія дала світу грецьких філософів. Фалес Мілетський у 6 ст. до н. е. ввів у науку метод математичних доведень і теорем. Зокрема, він довів рівність кутів рівнобедреного трикутника.

У тому ж сторіччі розвилась школа Піфагора. Піфагорійці розглядали парні й непарні числа, досліджували прості числа, намагались довести теорему ірраціональності числа $\sqrt{N}$, де N не є квадратом. Ввели досконалі числа — числа, які дорівнюють сумі своїх дільників, як 6 = 1+2+3. Довели формули:

$$1+3+5+\ldots+(2n-1) = n^2 \text{ та}$$
$$1^3 + 2^3 + \ldots n^3 = (n(n-1)/2)^2.$$

Доведена геометрично теорема Піфагора.

Отже, математика як наука зародилася в Стародавній Греції у 6 ст. до н. е.

Платон у 4 ст. до н. е. був енциклопедистом. Його заслугою є розвиток загальних принципів дедуктивного доведення.

Його учень Арістотель розвинув схему дедуктивних (вивідних) наук. Згідно з нею, основою науки є *визначення* основних понять і *твердження*, які описують властивості цих понять, зміст науки складають ланцюжки виразів, які виводяться один з одного.

Для цього застосовується ***дедукція*** — процес виведення правильного висновку, що гарантовано випливає, якщо початкові припущення істинні, а виведення базується винятково на основі попередньо наведених доведень і до процесу не додається нова інформація про предмет, що досліджується.

За Арістотелем, основні поняття мають бути інтуїтивно зрозумілі до такої міри, що для них не потрібне визначення і їх має бути достатньо, щоб із них можна було вивести решту понять даної науки. Твердження поділяються на такі, що виводяться (***теореми***) і такі, що не потребують

доведень, тобто початкові (***аксіоми***). Аксіоми повинні бути очевидними, щоб не потребувати доведення.

Після Арістотеля були Евклід, Архімед та інші. До 20 ст. найбільш відомим алгоритмом був алгоритм Евкліда, для знаходження найбільшого спільного дільника (НСД):

```
функція НСД(a, b)
    якщо a = 0
       повернути b
    поки b ≠ 0 виконувати
        якщо a > b
           a := a − b
        інакше
           b := b − a
    повернути a
```

Діофанта часто згадують в історії математики, як найвидатнішого алгебраїста грецького походження, або навіть як батька алгебри. У багатотомній праці «Арифметика» він абстрагувався від чисел як значень геометричних вимірів і показав, як розв'язувати деякі рівняння з цілими числами, які зараз називаються діофантовими.

Характерним для грецької математики були високий рівень абстракції та бездоганність дедукції. Багато сторіч (два тисячоліття) наукою вважалась лише наука за Арістотелем — така, яка ґрунтується на аксіомах і теоремах.

Лейбніц у 17 сторіччі запропонував ідею формалізації математики — створення «логічного числення», тобто певної універсальної математичної мови, якою в символічному вигляді можна записати процес доведення без неточностей та двозначностей.

Ця ідея певною мірою була реалізована Дж. Булем, де Морганом, Ч. Пірсом у другій половині 19 ст. Остаточно математична логіка як наука була закріплена Д. Гільбертом.

Завдяки Н. І. Лобачевському, К. Ф. Гаусу з'явилась неевклідова геометрія. Вона дала поштовх до розвитку математики різноманітних геометрій, алгебр в яких аксіоми задані не апріорно, а самими математиками.

На цьому тлі, на межі 19 і 20 ст. Д. Пеано, М. Пієрі та Д. Гільберт переосмислили арістотелевський підхід до математики з його інтуїтивною ясністю доведення теорем та очевидністю аксіом і затвердили концепцію «гільбертового формалізму».

Нарешті, у 20-х роках 20 ст. Л. Е. Я. Брауером і Г. Вейлем запропоновано самі правила доведення (висновування) теорем вважати за аксіоми. Отже, у певній математичній теорії стало можливим не тільки вибирати аксіоми, але й логіку доведення, яка відрізняється від арістотелевої. Таким чином, сучасна математична теорія ґрунтується на базисі затверджених аксіом та має множину теорем, які одержані за допомогою прийнятої логіки. Таку теорію можна зобразити так, як на рис. 2.

Слід зауважити, що ***теорія моделей*** у дискретній математиці вивчає загальні математичні (алгебраїчні) системи, які задані властивими їм аксіомами та правилами виводу.

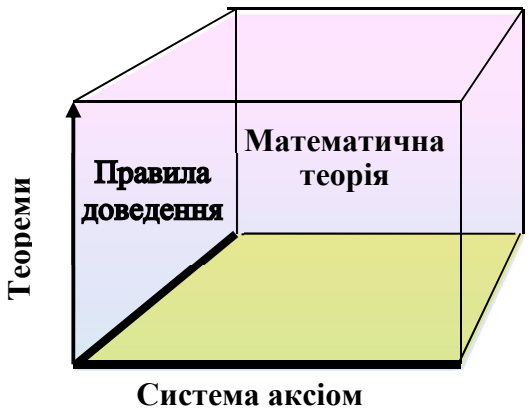


Рис. 2. Структура математичної теорії

Питання та завдання

1. Виконайте вручну алгоритм Евкліда для $a = 231$ та $b = 182$.
2. Виконайте вручну алгоритм розрахунку $1+3+5+\ldots+(2n-1) = n^2$ для $n = 16$.
3. Розв'яжіть діофантове рівняння $x^2 + y^2 = 100$.
4. Розв'яжіть діофантове рівняння $x^2 - y^2 = 4$.
5. Знайти $\sqrt{a}$ за алгоритмом Герона: $x_1 = 1$; $x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right) \approx \sqrt{a}$, $n = 1,2,\ldots$ для $a = 256$ з використанням раціональних чисел.
6. Знайти $\sqrt{a}$ за давньоіндійським алгоритмом: $x_1 \approx \sqrt{a}$ (перше наближення); $q_n = \frac{a - x_n^2}{2x_n}$, $x_{n+1} = x_n + q_n - \frac{q_n^2}{2(x_n + q_n)} = \sqrt{a}$, $n = 1, 2$ для $a = 16384$ з використанням раціональних чисел.
7. Довести графічно теорему Піфагора.
8. Розрізати тупокутний трикутник на кілька гострокутних трикутників.

1. Основи дискретної математики

1.1. Об’єкти

Людина мислить поняттями. Математика є системою різноманітних понять. Різні поняття узагальнюють думки про певні конкретні об’єкти. На Рис. 1.1 показано, як формується поняття та як воно відноситься до інших категорій, через які воно визначається та сприймається. Тут:

Сутність, суть — притаманна, виявлена, постійна і повторювана властивість об’єкта, яку можна зафіксувати та відстежити раціональним чином.

Поняття — думка або система думок, що виділяє й узагальнює об’єкти за загальними для даного класу та специфічними для них ознаками, сутностями.

Визначення — введення нового поняття чи об’єкта в міркування через комбінації або уточнення елементарних або раніше визначених понять.

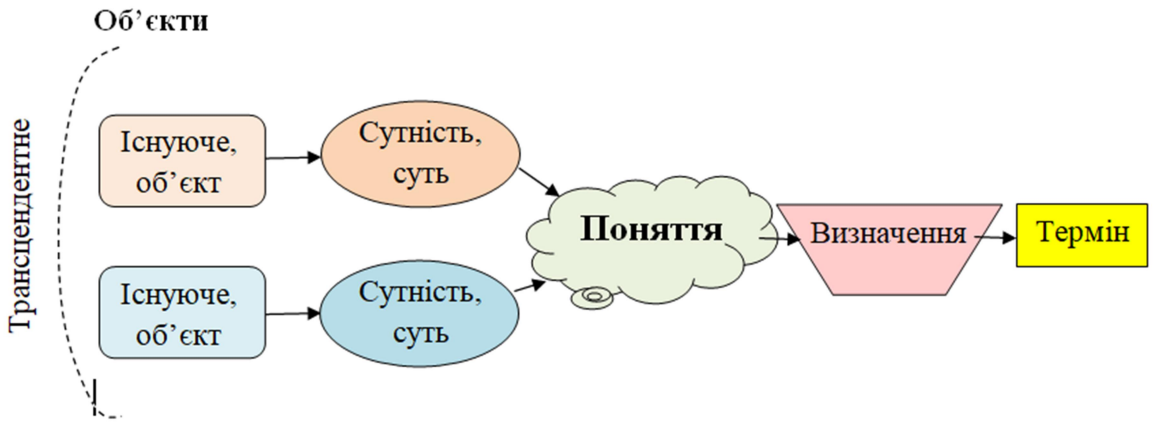


Рис. 1.1. Формування поняття та його відношення до інших категорій

Визначення в науці бувають прямі (конструктивні) та непрямі (дескриптивні).

Пряме визначення об'єкта — це явний опис об'єкта.

Наприклад, арифметичною прогресією називається ряд чисел a, $a + d$, $a + 2d$, … , де a і d — фіксовані числа.

Кілограмом є маса еталону кілограма.

Пряме або конструктивне визначення об'єкту є одночасно й доведенням його існування.

Непряме визначення об'єкта — опис, який задає об'єкт як перелік його необхідних властивостей.

Наприклад, арифметична прогресія — це послідовність чисел a_1, a_2, a_3, … для якої різниця $c = a_{i+1} - a_i$ стала для всіх $i = 1, 2, \ldots$.

Кілограм — це маса твердого тіла вагою 9,8067 Н.

Непряме визначення об'єкта не доводить його існування. Наприклад, вічний двигун — це механізм, який виконує роботу без підведення енергії.

Термін — визначення об'єкта, поняття з жорсткими границями його сутностей, яке є його назвою у вигляді символа, слова чи словосполучення.

Наприклад, термін у вигляді символа: двійкова цифра — це символ 0 чи 1, число пі — це символ π.

Трансцендентне — це те, що не можна відокремити, відчути, зафіксувати у практичній діяльності.

Знання — сукупність понять разом з відношеннями між ними, які ґрунтуються на певній моделі світу і які можна застосувати у практичній діяльності.

Об'єкти в математиці — це об'єкти різної природи, що відрізняються один від одного найменуваннями, які їм умовно приписують.

У дискретній математиці, у тому числі в дискретних структурах, розглядаються лише конструктивні об’єкти.

Конструктивні об’єкти мають дві основні властивості:

— мають пряме визначення;

— повинні чітко, безпомилково відрізнятись один від одного.

Оскільки об’єкт має пряме, тобто конструктивне визначення, то він завжди існує або може бути побудований (сконструйований) за цим визначенням. Наприклад, будь-яке багаторозрядне просте число може бути знайдене за алгоритмом решета Ератосфена.

Вимога чіткого виділення даного об’єкта серед інших є основою дискретної математики. Вона дає змогу формально, а не інтуїтивно, виділяти об’єкти та оперувати з ними.

Бувають ***складні конструктивні об’єкти***, які компонуються з простіших об’єктів за прямим визначенням. Це, наприклад, слова з літер, багаторозрядні числа з цифр, масиви, матриці даних.

Якщо конструктивний об’єкт складний, то він, як правило, має певний «початок координат», за яким його можна чітко відрізнити серед інших об’єктів та виділити його елементи. Наприклад, це перша буква слова, перший елемент першого рядка матриці тощо.

На відміну від дискретної математики, у математичному аналізі розглядаються нескінченно малі та нескінченно великі об’єкти, які не є конструктивними. Наприклад, два дійсні числа, що відрізняються між собою на нескінченно малу величину, неможливо ні зобразити конкретно, ні розрізнити одне від одного. Зате, на їх основі можна визначити поняття похідної. Трансцендентні числа тому й так називаються, бо таке число неможливо зобразити точно, використовуючи якісь конструктивні об’єкти.

Поняття конструктивного об’єкту розвив А. А. Марков (мол.) у 50 — 60-х роках 20 ст. у рамках теорії конструктивної математики. Метою

розвитку цієї теорії було створення містка між класичним матаналізом та обчислювальною математикою і теорією алгоритмів.

1.2. Множини

Множина — це сукупність певних об'єктів довільної природи, які мають якусь спільну сутність. Наприклад, можна говорити про множину всіх книг у певній бібліотеці, множину літер українського алфавіту. Об'єкти, які складають множину, називають ***елементами*** цієї множини.

Множина задається декларативно (конструктивно) як перелік її елементів. Нехай множина X містить елементи $a, b, c, \ldots, k$. Для запису цього факту застосовують такий вираз:

$$X = \{a, b, c, \ldots, k\}.$$

Факт належності якогось елемента k або неналежності елемента p до множини X позначають за допомогою логічних операторів належності «$\in$» та неналежності «$\notin$» так:

$$k \in X;\ p \notin X.$$

Також множина задається непрямим способом із вказівкою властивостей елементів. Наприклад, множина натуральних чисел, менших за 100, позначається як

$$A = \{x \in \mathbb{N} \mid x < 100\}.$$

Тут літера $\mathbb{N}$ — це загальноприйняте позначення множини натуральних чисел, вертикальною рискою відділяється частина, де перелічуються властивості елементів.

Підмножиною B множини A називається множина, усі елементи якої належать до множини A, хоча можуть бути елементи з A, які не належать B. Це позначається логічним оператором включення «$\subset$» як $B \subset A$, тобто

$$B = \{x \in A\}.$$

Потужністю $|A|$ множини A називається число її членів.

Поняття множини вводиться аксіоматично і тому множину не можна означити, застосовуючи інші означені поняття.

Якщо можна класифікувати конструктивні об'єкти за загальним правилом їх конструювання, то множину таких об'єктів називають ***ансамблем*** конструктивних об'єктів.

Типовим прикладом ансамбля конструктивних об'єктів є множина слів у даному алфавіті. При цьому правило чи спосіб конструювання полягає в побудові скінченних ланцюжків літер, тобто слів.

Множини натуральних $\mathbb{N}$, раціональних $\mathbb{R}$ чисел є ансамблями, бо є правило їх побудови.

1.3. Математичні структури

Отже, як було показано вище, математичну теорію можна зобразити як деякий простір, що складається з аксіом, правил доведення (також аксіом) та теорем, які визначені на їх основі (Рис. 2). Така теорія оперує з визначеними математичними об'єктами. Залишилось визначити взаємовідношення між цими об'єктами, аксіомами та теоремами.

З метою розвитку універсального підходу до вивчення математики, Ніколя́ Бурбакі у 1948 р. ввів поняття математичної структури.

Математична структура — це визначені одна чи кілька множин об'єктів із заданою системою відношень між елементами цих множин. Причому об'єкти можуть бути невизначеними, мати різну природу і відрізнятись один від одного лише за найменуванням, яке надається умовно.

Наприклад, ***структура натуральних чисел*** визначається як множина $\mathbb{N}$ елементів, що називаються числами, у якій визначені дві бінарні операції, що називаються додаванням та множенням чисел. Ці

операції зіставляють з кожними двома числами m і n третє число, яке позначається відповідно через $m + n$ та через $m \cdot n$ або mn. Крім того, у множині $\mathbb{N}$ виділяється «найперше» число 1 та задається унарна операція «’» переходу до наступного числа $n' = n + 1$. У зв’язку з цим структура натуральних чисел **N** позначається як

$$\mathbf{N} = \langle \mathbb{N}; +, \cdot, ' \rangle.$$

У цій структурі операції додавання та множення визначені непрямими визначеннями, тобто, дескриптивно через списки аксіом операцій, які задають їхні властивості. Це відомості з підручників з арифметики.

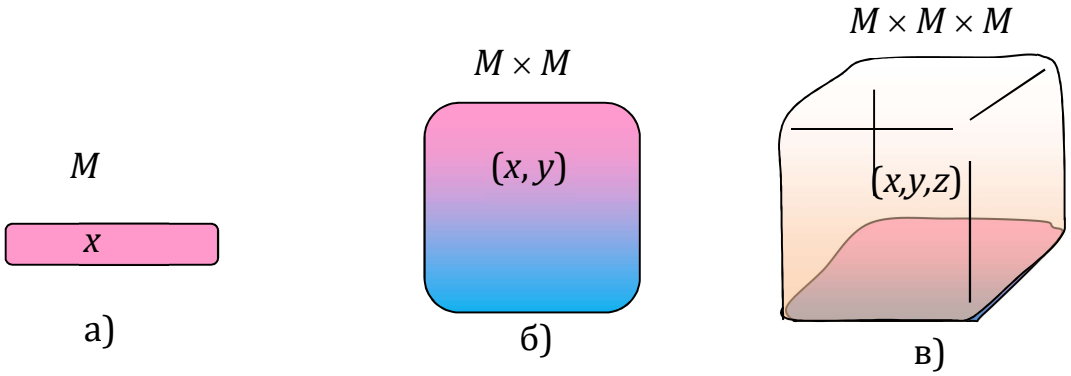
1.4. Відношення

У математичних структурах задаються унарні, бінарні, тернарні і т. д. відношення на певній множині M. ***Унарне відношення*** — це просто виділення якоїсь підмножини в M.

Бінарне відношення R — пов’язує пари елементів з M. Причому вказуються усі пари (x, y), $x, y \in M$, для яких існує відношення $x\,R\,y$ або $R(x, y)$. Отже, бінарне відношення вказує певну підмножину в множині пар $(x, y) \in M \times M$. Тут $M \times M = M^2$ — декартів добуток (декартів квадрат) множин M, який являє собою сукупність усіх можливих пар (x, y) (Рис. 1.2). ***Декартів добуток*** визначається наступним чином:

$$A \times B = \{(a, b) \mid a \in A, b \in B\}.$$

Нехай маємо множину $M = \{0,1,2,3\}$ і відношення на ній $x\,R\,y$, яке означає, що x більший за y, тобто $x > y$. Тоді множину M^2 і відношення R можна зобразити графічно так, як на Рис. 1.3.

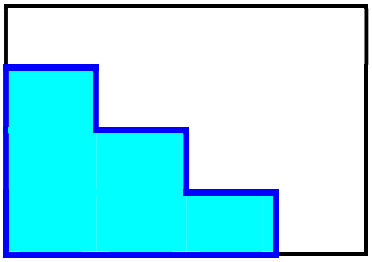


а) б) в)

Рис. 1.2. Множина M (а), її декартів добуток — квадрат (б) та куб (в)

	y=0	1	2	3
x = 0	(0,0)	(0,1)	(0,2)	(0,3)
1	(1,0)	(1,1)	(1,2)	(1,3)
2	(2,0)	(2,1)	(2,2)	(2,3)
3	(3,0)	(3,1)	(3,2)	(3,3)

а)



б)

Рис. 1.3. Декартів квадрат M^2 (а), та відношення $x > y$ на ньому (б)

У загальному випадку, бінарне відношення між двома множинами A та B є підмножиною від $A \times B$. Назви ***відповідність*** та ***двомісне відношення*** є синонімами бінарного відношення.

Тернарне відношення — пов'язує трійки (x, y, z) елементів з M і задається як певна підмножина декартового куба $M \times M \times M$ (Рис. 1.2, в).

Аналогічно ***m-арне відношення*** R визначається як підмножина декартового степеня

$$M \times M \times \ldots \times M.$$

Тоді запис $R(x_1, x_1, \ldots, x_m)$ означає, що елементи $x_1, x_1, \ldots, x_m \in M$ пов'язані відношенням R.

Формально ***математична структура*** визначається як система

$$\mathbf{S} = \langle M; R_1, R_2, \ldots, R_k \rangle,$$

яка складається з основної множини $M = \{a, b, c, \ldots\}$ та заданих на цій множині відношень $R_1, R_2, \ldots, R_k$, які є унарними, бінарними і т. д. Основні

властивості відношень R_i, $i = 1, 2, \ldots, k$, задаються аксіомами, які також мають бути включені до опису структури. Розвиток певної математичної теорії полягає у дослідженні інших властивостей структури **S**, яке зводиться до складання і доведення різних теорем на основі наявних аксіом і властивостей відношень R_i.

1.5. Деякі важливі властивості відношень

Обернене до R відношення R^{-1} — це таке, що для будь-яких x, y випливає, що якщо є $x\,R\,y$, то є відношення $y\,R^{-1}\,x$. Наприклад, для відношення «більше або дорівнює» оберненим є відношення «менше або дорівнює», як зображено кольором на Рис. 1.4.

Протилежне відношення $\bar{R}$ існує тоді, коли не існує відношення R. Так, для відношення «більше або дорівнює» протилежним є «менше» (Рис. 1.4).

Композицією відношень R_1 і R_2 на множині M (позначають $R_1 \circ R_2$) називають таке відношення R на $M \times M$, що $x\,R\,y$ тоді й тільки тоді, коли існує елемент $z \in M$, для якого виконується $x\,R_1\,z$ і $z\,R_2\,y$.

На Рис. 1.4 показана композиція відношення $x\,R\,y = x \geq y$ та R^{-1}. Наприклад, композицією відношень R_1 — є син і R_2 — є братом на множині чоловіків є відношення $R_1 \circ R_2$ — є небожем.

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

R а)

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

R^{-1} б)

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

$\bar{R}$ в)

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

$R \circ R^{-1}$ г)

Рис. 1.4. Відношення «більше або дорівнює» R для множини $M = \{0, 1, 2, 3\}$ та відповідні відношення R^{-1}, $\bar{R}$ та $R \circ R^{-1}$

1.6. Приклад математичної структури

Нехай є 4 учні *А, Б, В, Г,* тобто множина $B = \{А, Б, В, Г\}$. Учні мають певні родинні стосунки, які закріплені наступними аксіомами:

A_1: *А* є братом *В.*

A_2: *В* є братом *Г.*

A_3: *Б* не є братом *Г.*

Запишемо ці відношення за допомогою відношення спорідненості *P*. Для нього наявність запису *a P b* означає, що *a* і *b* — родичі.

Для відношення *P* є ***протилежне відношення*** $\overline{P}$. Тобто $x \overline{P} y$ означає, що *x* та *y* — не родичі.

Відношення *P* можна замінити на відповідну операцію $P(x, y)$, яка дорівнює 1, якщо відношення істинне і 0 — якщо хибне. Тут 1 і 0 належать до ***булевої множини*** $\{0, 1\}$. Отже, якщо $P(x, y) = 1$, то $\overline{P}(x, y) = 0$ і навпаки.

Тоді три аксіоми можна зобразити як

A_1: $A\ P\ B.$

A_2: $B\ P\ Г.$

A_3: $Б\ \overline{P}\ Г.$

З функцією *P* пов'язані ще дві аксіоми, які накладають на неї властивості дедукції.

Це аксіома ***симетричності*** *P*:

A_I: Якщо $x\ P\ y$ — правильне, то й $y\ P\ x$ — правильне; або $x\ P\ y \Rightarrow y\ P\ x$;

та аксіома ***транзитивності***:

A_{II}: Якщо правильні $x\ P\ y$ і $y\ P\ z$, то й $x\ P\ z$ — правильне; або $x\ P\ y \wedge y\ P\ z \Rightarrow x\ P\ z$.

Тут «$\Rightarrow$» — це логічний оператор ***імплікації***, який при $x \Rightarrow$ y означає, що з *x* випливає *y,* а знак «$\wedge$» означає ***логічний оператор І***.

Отже, маємо математичну структуру $\langle B; P\rangle$ і треба її дослідити. Множина B має чотири елементи. Відповідно, декартів добуток множини має $4 \cdot 4 = 16$ елементів, які являють собою впорядковані пари.

Метою цього простого дослідження є знаходження таких пар, для яких відношення P істинне. Оскільки відомі лише три відношення P у вигляді аксіом, залишилось формально довести $16 - 3 = 13$ теорем щодо інших відношень. За аксіомою A_I кількість пар різних людей, що розглядаються, скорочується удвічі. А таких пар, за формулою числа невпорядкованих пар, $4 \cdot 3 / 2 + 4 = 10$. Тут також слід урахувати те, що, у даному випадку, сам собі — не родич, тобто це може бути третя аксіома дедукції A_{III}: $x \bar{P} x$. З урахуванням цього, число аксіом і теорем скорочується до 6. Отже, треба довести лише 3 теореми.

Теорема 1. *А Р Г.*

Доведення. Підставивши елементи в аксіому A_{II}, одержимо:

$$A\ P\ B \wedge B\ P\ Г \Rightarrow A\ P\ Г.$$

Теорема 2. *Б* $\bar{P}$ *А.*

Доведення. Від супротивного: нехай *Б Р А,* тоді з *А Р Г* за аксіомою A_{II} випливає, що *Б Р Г,* а це — неправильно за аксіомою A_3. У формальному записі:

$$Б\ P\ A \wedge A\ P\ Г \Rightarrow Б\ P\ Г \neq A_3.$$

Теорема 3. *Б* $\bar{P}$ *В* — доводиться так само, як і теорема 2.

1.7. Властивості математичних структур

Для кожної математичної структури, яка задається через основні, аксіоматичні відношення R_i та аксіоми висновування, постають питання про несуперечливість, повноту та незалежність цієї аксіоматики.

Система аксіом називається ***несуперечливою,*** якщо з аксіом не можна вивести двох висновків, які взаємно виключають один одного.

Тобто на основі таких аксіом шляхом дедукції ми ніколи не прийдемо до протиріччя.

Несуперечливість визначається через моделювання чи інтерпретацію структури. При цьому до конкретних елементів структури прикладаються наявні аксіоми і перевіряється наявність чи відсутність протиріч. Очевидно, що немає сенсу розглядати суперечливі системи.

Система аксіом називається ***повною,*** якщо вона припускає єдину реалізацію. Тобто за повною системою аксіом не можна побудувати кілька різних математичних структур.

Тут постає питання ізоморфізму структур, оскільки повна система аксіом може породжувати ізоморфні структури. Дві структури $\boldsymbol{S} = \langle M; R_1, R_2, \ldots, R_k \rangle$ і $\boldsymbol{S'} = \langle M'; R'_1, R'_2, \ldots, R'_k \rangle$ є ***ізоморфними,*** якщо існує взаємно однозначна відповідність $M \leftrightarrow M'$, $R_1 \leftrightarrow R'_1$, …, $R_k \leftrightarrow R'_k$. Ізоморфні структури характеризуються тим, що кожному відношенню в одній структурі відповідає аналогічне відношення між відповідними елементами у другій структурі. Іншими словами, ізоморфні структури — це просто одна й та ж сама структура, у них лише є відмінності в різних назвах і позначках структурних елементів.

У математиці розглядають як повні, так і неповні системи аксіом. Багато алгебраїчних систем, таких як група, кільце, поле, решітка (розглядаються нижче), є неповними.

Система аксіом називається ***незалежною,*** якщо жодну з аксіом неможливо вивести з інших аксіом цієї системи, тобто довести як теорему. Тобто така система є мінімальною.

Звичайно, структура з незалежною системою аксіом є досконалою. Але іноді важко довести, що якась аксіома є насправді теоремою. З іншого боку, деякі теореми можуть бути дуже зручними або очевидними при побудові математичної теорії, і їх можуть додати до множини аксіом.

1.8. Операції

Алгебраїчна операція — це важливий окремий випадок відношення. ***Бінарна операція*** зіставляє два довільні, але впорядковані елементи $x, y \in M$ з певним третім елементом $z \in M$ — результатом застосування цієї операції до елементів *x, y.* Якщо операцію позначити як «⁕» або літерою *f,* то така операція записується як

$$z = x \ast y; \text{ або } z = f(x, y); \text{ або } (x, y) \xrightarrow{f} z.$$

Впорядкованість операндів x, y означає, що в загальному випадку операція — несиметрична або ***некомутативна***, тобто $x \ast y \neq y \ast x$.

Очевидно, що бінарна операція — це спеціальний тип тернарного відношення, яке пов'язує три елементи *x, y* та *z* між собою. Таке відношення *R* характеризується тим, що для кожної пари елементів *x, y* з *M* існує *єдиний* елемент $z \in M$, для якого наявне сполучне відношення *R*. Формально це записується так:

$$\forall\, x, y \in M\ \exists!\, z\ (z \in M \mid R(x, y, z)). \qquad (1.1)$$

Тут ∀ — ***квантор загальності***, який читається як «для всіх», оголошує, що маються на увазі будь-які елементи *x, y* зі вказаними властивостями, тут — ті, що належать до множини *M*. ∃ — це ***квантор існування***, який інтерпретується як «існує», «є принаймні один» або «для деяких». У цій формулі він уточнюється як ∃!, що означає «існує та є лише єдиний». Варто зауважити, що дані квантори та інші символи-терміни ввів у математичну практику Н. Бурбакі.

Вирази в дужках (1.1) уточнюють властивості елемента, виділеного квантором існування, і перелічені через відокремлюючу риску «|». Тут цей вираз читається як: «існує таке *z,* яке належить до *M* та існує відношення $R(x, y, z)$».

Прикладами операцій є арифметичні операції над множиною цілих чисел $\mathbb{Z}$.

Крім того, елементи x, y та z можуть належати різним множинам. Наприклад, операція $\boldsymbol{a} = f(a, b)$ відносить дві точки $a, b \in N$, що належать множині точок, до вектора $\boldsymbol{a} = \overrightarrow{ab} \in V$, який належить до множини векторів. Кажуть, що ця операція виконує ***відображення*** декартового квадрату $N \times N$ у простір векторів V, що записується як $N \times N \longrightarrow V$.

Аналогічно розглядається ***n-арна операція,*** що задана на множині M як n+1–арне відношення R, за яким кожним впорядкованим n елементам x_1, x_1,..., x_n ставиться у відповідність єдиний елемент z тієї самої або іншої множини.

1.9. Алгебраїчні структури

Алгебраїчною структурою називається структура з певною множиною M, невеликою кількістю бінарних алгебраїчних операцій та такими аксіомами, як асоціативність, комутативність, дистрибутивність.

Порядком структури називають потужність її носія — $|M|$.

У алгебраїчних структурах розрізняють операції типу множення «*» та типу додавання «+». Найчастіше, ці операції не співпадають з операціями арифметичного множення та додавання, хоча операцію «*» називають добутком, а «+» — сумою.

Для операцій типу множення завжди знайдеться ***нейтральний елемент*** e**,** множення на який залишає результат незмінним, тобто $\exists\, e \in M\ (x \in M;\ x * e = x)$. Аналогічний елемент є для операції додавання, $\exists\, e \in M\ (x \in M; x + e = x)$.

Структури з такими операціями називають ***мультиплікативними*** та ***адитивними***, відповідно.

Комутативна структура — це така, яка має аксіому комутативності, тобто,

$$\forall\, a, b \in M\ (a * b = b * a).$$

Асоціативна структура — це така, яка має аксіому асоціативності, тобто,

$$\forall\, a, b, c \in M\ \big((a * b) * c = a * (b * c)\big).$$

Моноїд — це асоціативна структура, яка має одну операцію типу множення та нейтральний елемент.

Моноїдом може бути множина натуральних чисел з операціями додавання чи множення. При додаванні нейтральним елементом є нуль, а при множенні — одиниця.

Важливою властивістю моноїда, як і будь-якої структури є та, що результат операції з будь-якими елементами повинен належати до тієї самої множини. Наприклад, структура парних чисел з операцією «+» є моноїдом, а непарних чисел — ні.

Незвичним моноїдом є множина всіх слів з алфавітом, наприклад, $\{a, b, c\}$ з операцією конкатенації «+». Для нього справджується асоціативність, як наприклад,

$$(aba + baba) + caca = aba + (baba + caca) = abababacaca.$$

Група — це алгебраїчна структура $\boldsymbol{G} = \langle G;\ \oplus\rangle$ з наступними аксіомами:

Γ_1: $\forall\, a, b, c \in G\ \big((a \oplus b) \oplus c = a \oplus (b \oplus c)\big)$,

тобто групова операція є асоціативною.

Γ_2: $\exists\, e \in G\ (x \in G;\ x \oplus e = e \oplus x = x)$, у групі є одиничний елемент.

Γ_3: $\forall\ a \in G\ \exists\, a^{-1} \in G\ (a \oplus a^{-1} = a^{-1} \oplus a = e)$, тобто для кожного елемента a групи знайдеться йому ***обернений елемент*** a^{-1}.

Абелева або ***комутативна група*** — це група, яка має додатково аксіому комутативності.

Скінченна група — це група, множина якої G є скінченною. Відповідно, якщо множина G є нескінченною, то і група є ***нескінченною***. Кількість елементів |G| скінченної групи є її ***порядком***.

Прикладом групи є множина раціональних чисел з операцією множення «*». У ній нейтральний елемент $e = 1$. Для неї для дробу виду a/b завжди знайдеться обернений елемент b/a. І це є абелева група, бо

$$\frac{a}{b} * \frac{c}{d} = \frac{c}{d} * \frac{a}{b}.$$

Ця група є нескінченною, бо повинен знайтись результат операції для будь-яких великих чисел.

Щоб множина чисел G була скінченною, часто використовують операцію взяття за модулем порядку групи $p = |G|$. Наприклад, якщо $p = 10$, то результат добутку елементів 7 та 7 за модулем p буде $7 \otimes 7 = 9$. Так створюються, так звані, групи ***модульної арифметики.***

Наприклад, 8-бітові числа в мікропроцесорах та операції додавання чи множення формують групи арифметики за модулем $2^8 = 256$.

Група підстановок — це скінченна група порядку n, у множину G якої входять n операцій підстановки елементів іншої множини M. Нехай $|M| = 5$. Тоді операцією підстановки і власне, елементом множини G є матриця типу: $\begin{pmatrix} 1\,2\,3\,4\,5 \\ 3\,5\,4\,1\,2 \end{pmatrix}$, яка показує, як слід переставити елементи множини M. Тут перший елемент переставляється на третє місце, другий — на п’яте і т.д. Одиничним елементом є матриця $\begin{pmatrix} 1\,2\,3\,4\,5 \\ 1\,2\,3\,4\,5 \end{pmatrix}$, яка фактично не виконує перестановку.

Множенням двох підстановок є нова підстановка після їх послідовного виконання, яка належить **G**. Наприклад, множення однієї підстановки на обернену дає одиничний елемент:

$$\begin{pmatrix} 1\,2\,3\,4\,5 \\ 3\,5\,4\,1\,2 \end{pmatrix} \star \begin{pmatrix} 1\,2\,3\,4\,5 \\ 4\,5\,1\,3\,2 \end{pmatrix} = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 1\,2\,3\,4\,5 \end{pmatrix}.$$

Аналогічні властивості має група поворотів вектора на *n* різних кутів.

Кільце — це алгебраїчна структура, у якій визначені дві асоціативні операції — додавання і множення, причому за додаванням кільце є комутативною групою, а за множенням — воно дистрибутивне відносно додавання. Операцію множення, як правило, позначають точкою. Тобто кільце — це структура

$$\boldsymbol{K} = \langle M; \cdot, + \rangle,$$

для якої виконуються аксіоми:

К$_1$: $\forall\, a, b, c \in M \left((a + b) + c = a + (b + c)\right)$.

К$_2$: $\forall\, a, b \in M \left(a + b = b + a\right)$.

К$_3$: $\exists\, 0 \in M \;\forall\, a \in M \left(a + 0 = a\right)$.

К$_4$: $\exists\, 0 \in M \;\forall\, a \in M \left(a + (-a) = 0\right)$.

К$_5$: $\forall\, a, b, c \in M \left((a \cdot b) \cdot c = a \cdot (b \cdot c)\right)$.

К$_6$: $\forall\, a, b, c \in M \left((a + b) \cdot c = a \cdot c + b \cdot c;\; a \cdot (b + c) = a \cdot b + a \cdot c\right)$.

Комутативне кільце — це кільце, у якого є ще аксіома комутативності множення:

К$_7$: $\forall\, a, b \in M \left(a \cdot b = b \cdot a\right)$.

***Унітарне комутативне кільце* (*кільце з одиницею*)** — це комутативне кільце, у якого є ще аксіома множення на одиницю:

$К_8$: $\exists 1 \in M \; \forall \; a \in M \; (a \cdot 1 = 1 \cdot a = a)$.

Основним прикладом унітарного комутативного кільця є кільце цілих чисел.

На жаль, кільце не можна вважати мультиплікативною групою, бо для елемента 0 немає йому оберненого елемента. Щоби перетворити кільце в мультиплікативну групу, слід додати ще одну аксіому або теорему:

$Т_8$: $\forall \; a \in M, a \neq 0 \; \exists \; a^{-1} \in M \; (a \cdot a^{-1} = 1)$.

Комутативне поле — це комутативне кільце, для якого також виконується аксіома $Т_8$.

Отже, у поля є дві операції: додавання та множення, причому всі елементи поля за додаванням і ненульові елементи за множенням формують комутативні групи, а операції додавання і множення пов'язані між собою дистрибутивним законом.

Типовими прикладами комутативних полів є поле раціональних чисел і поле дійсних чисел. Серед скінченних комутативних полів важливе місце займають поля Галуа.

1.10. Графи

Задачі з теорії графів розв'язував Ейлер ще у 18 ст. Але термін «граф» вперше з'явився в книзі угорського математика Д. Кьоніга у 1936 р.

Графом (***неорієнтованим графом***) називається пара $G = (V, E)$, де V — непуста множина, $E \subset V \times V$.

Елементи $v_i \in V$ називаються ***вершинами*** (vertexes), а елементи $e_j \in E$ — ***ребрами*** (edges). Число вершин графа $n = |V|$ називається його

порядком. Якщо у графа $m = |E|$ вершин, то G називають (n, m)-***графом.*** Отже, $i = 1, 2,\ldots, n, j = 1, 2,\ldots, m.$

Вершини u і v графа є ***суміжними,*** якщо $(u, v) \in E$, та є ***несуміжними***, якщо $(u, v) \notin E.$ Якщо $e = (u, v) \in E,$ то вершини u і v називають його кінцями. Кажуть, що ребро e ***з'єднує*** вершини u і $v.$

Вершина v і ребро e є ***інцидентними,*** якщо $e = (u, v)$, тобто, якщо v є кінцем ребра, і ***неінцидентними*** в інакшому випадку.

Формально розглядаючи, граф — це математична структура $\boldsymbol{G} = \langle \boldsymbol{V}; R\rangle$, де $\boldsymbol{V} = V$, а R — бінарне відношення, яке задає суміжність вершин $v_i \in \boldsymbol{V}$, тобто $u\ R\ v$ фактично позначає ребро (u, v).

Множини $V,\ E$ зручно задавати як множину номерів вершин V_G і множину пар номерів суміжних вершин E_G.

Приклад. Певний (5, 6)-граф задається як

$$V_G = \{1, 2, 3, 4, 5\},\ E_G = \{\{1, 2\}, \{1, 3\}, \{2, 5\}, \{3, 4\}, \{3, 5\}\}.$$

Часто зустрічається подання графа квадратною матрицею суміжності, у клітинці i, j якої стоїть 1, якщо вершини v_i, v_j суміжні, і 0 — якщо несуміжні. Для даного прикладу матриця суміжності виглядає так:

$$E = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

За матрицею добре видно, що вершини 1 і 2 суміжні, а вершини 1 і 4 — несуміжні.

Графи зручно зображати у вигляді рисунків. Наприклад, на Рис. 1.5 показаний (5, 6)-граф, заданий вище.

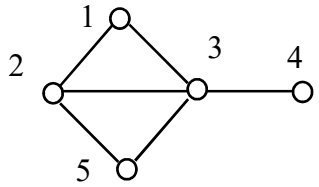


Рис. 1.5. Граф

З рисунку графу одразу видно, що, наприклад, вершини 1 і 2 суміжні, а вершини 1 і 4 — несуміжні, а також те, що вершина 1 інцидентна ребру {1, 3}.

Граф — це зручна модель для зображення різноманітних відношень, які виникають у людській практиці. Наприклад, структура $\langle B; P \rangle$, яка розглядалась вище при вивченні родинних відношень між учнями, виражається як граф на Рис. 1.6.

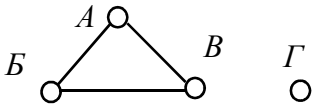


Рис. 1.6. Граф структури

При цьому відношення між елементами *А, Б, В, Г* є симетричними або комутативними: якщо *А* брат *Б,* то *Б* — брат *А,* що зумовлює використання неорієнтованих ребер. Але, якщо ми розглядаємо такі відношення, як відношення порядку, наприклад, хто старший, то властивостей неорієнтованого графу недостатньо. Для цього більш підходящий орієнтований граф.

1.11. Алгоритми

1.11.1. Алгоритми і моделі для їх подання

Під алгоритмом зазвичай мають на увазі певний метод покрокового перетворення початкових даних в очікувані результати. До 16 ст. твори з мистецтва розрахунків називались алгоритмами. До 20 ст. слово «алгоритм» найчастіше асоціювалось лише з алгоритмом Евкліда. А якісь алгоритми, наприклад, розв'язання рівняння, знаходження числової функції, називались методами. Отже, алгоритми, які ми маємо на увазі, у людській практиці з'явились так само давно, як і математика.

Визначення алгоритму як ефективної обчислювальної процедури та поняття функції, що може бути обчислена, з'явились у роботах Бореля (1912 р.) та Вейля (1921 р.).

У 1936 р. А. Тюрінг та Е. Пост незалежно один від одного запропонували модель для подання алгоритмів, яка знана як машина Тюрінга. Ця абстрактна модель складається з нескінченної стрічки, яка в своїх комірках зберігає символи, такі як 0 чи 1, голівки запису-читання цих комірок та керуючого пристрою на основі таблиці переходів (Рис.1.7). У кожен момент часу модель знаходиться в певному стані, який названо конфігурацією машини. Процес обчислення заданої функції полягає у встановленні початкової конфігурації S_0 (вхідні дані) та послідовній зміні проміжних конфігурацій S_i аж до зупинки машини з кінцевою конфігурацією S_n, яка є результатом.

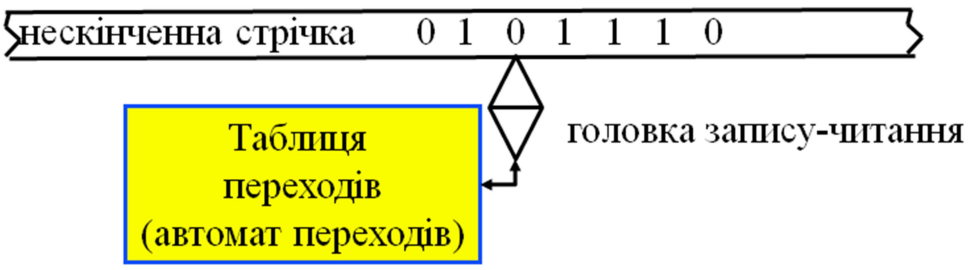


Рис. 1.7. Машина Тюрінга

Черч і Тюрінг шукали відповідь для задачі розв'язності, яка зводиться до пошуку алгоритму, що вирішує, чи дасть відповідь даний обчислювальний алгоритм, чи ні. Вони застосовували формалізацію поняття обчислювальної функції. Зокрема, Тюрінг висунув тезу про те, що кожна обчислювальна функція обчислюється машиною Тюрінга. А Черч у своїй тезі наголошував, що така функція є частково-рекурсивною функцією.

Основною думкою Тюрінга була та, що ***алгоритм*** — це обчислювальний процес, який задано на певній моделі комп'ютера, що описана за допомогою точних математичних понять [2]. У випадку Тюрінга така модель це машина Тюрінга, а у випадку Черча — схема підстановки частково-рекурсивних функцій.

Поняття алгоритму належить до таких фундаментальних понять, які не можуть бути точно виражені через інші поняття. Таке визначення, як «***Алгоритм*** — це точний припис, який задає обчислювальний процес, що починається з довільних вхідних даних, і спрямований на одержання результату, який повністю визначається цими вхідними даними» (Марков мол.) дає змогу уявити алгоритм як обчислювальний процес, що складається з послідовності станів деякої обчислювальної моделі.

Проте, для певного наукового напрямку можна формально визначити алгоритм за допомогою опису моделі, яка його задає та виконує. Причому кожній такій моделі притаманний відповідний спосіб подання алгоритму. Розглянемо коротко кілька найбільш поширених моделей.

Колмогоров з Успенським у 1958 запропонували ***модель колмогоровського типу*** (KU machine). У ній, крім того, що алгоритм для будь-яких даних із заданої множини дає результат і є покроковим, є важливі уточнення. По-перше, кроки повинні бути обмеженої складності й обробляти інформацію одержану з попереднього кроку. По-друге, інформація, яка береться для обробки на черговому кроці, має бути локаль-

ною, наперед обмеженою межами активної частини даних. Таке уточнення про складність кроків та локальність даних має слушне тлумачення:

— елементарні операції з фіксованою складністю дають змогу ввести метрику — ***складність алгоритму*** як підрахунок кількості операцій, які виконав алгоритм (***часова складність***) та кількість комірок пам'яті, які для цього необхідні (***просторова складність***);

— локальність дії операцій є передумовою для складання таких алгоритмів, як цикли з керуванням, наприклад, за лічильником ітерацій, алгоритмів з викликами підпрограм, рекурсивні алгоритми;

— модель є наближеною до реальних електронних комп'ютерів у тому, що в них також виконуються елементарні операції за простими командами за обмежений часовий період і в кожен момент часу виконується доступ до даних, які беруться з комірок пам'яті, що є геометрично близькими.

Правда, «обмежена складність» та «локальність інформації» є інтуїтивними поняттями і обмежують формалізм визначення алгоритму.

Вказана вище машина Тюрінга також є моделлю колмогоровського типу.

У 1954 р. Марков запропонував модель для подання нормальних алгоритмів Маркова. Ця модель не має властивості локальності даних. Але Марков запропонував вважати дані, що обробляються, конструктивними об'єктами. Тоді, врешті-решт, конфігурацію машини Тюрінга можна вважати певним складним конструктивним об'єктом. Як результат, обчислювальний процес алгоритму — це процес зміни конструктивних об'єктів-станів, починаючи з початкового S_0 та закінчуючи результуючим S_n.

Крім того, машина Маркова зображується машиною Колмогорова, якщо кроки з нелокальними даними замінити на ланцюжки кроків з локальними даними.

Отже, алгоритм як обчислювальний процес можна зобразити таким, як на Рис. 1.8.

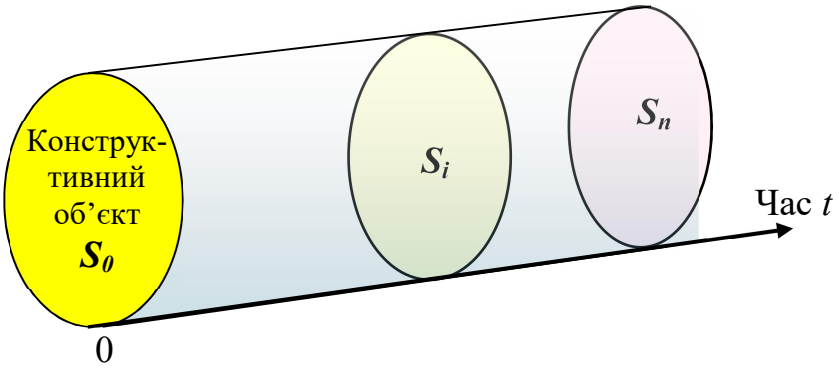


Рис. 1.8. Обчислювальний процес у моделі колмогоровського типу

Машина Тюрінга — це модель для виконання найбільш узагальнених і абстрактних алгоритмів. Але на практиці важливо вивчати алгоритми, які розподілені на певні класи. Для вивчення алгоритмів, які є більш пристосованими до практики, був розроблений ряд різних моделей.

У 1968 р. Д. Кнут запропонував модель машини зі вказівниками, яка краще відображає особливості програмування з неявною адресацією. У своїй фундаментальній праці «Мистецтво програмування» від запропонував модель MIX, яка відображає основні ознаки комп'ютерних архітектур 60 — 70-х років. Модель використовувалась як для подання алгоритмів, що досліджувались, так і для уточненого підрахунку їх складності.

У 1970 р. Шьонхаге запропонував модель машини з пам'яттю, що модифікується (storage modification machine, SMM), яка є вдосконаленням машини Колмогорова-Успенського і спрямована на врахування непрямої адресації в алгоритмах.

Кук і Рекхоу в 1973 р. запропонували, а Ахо, Ульман і Хопкрофт у 1974 р. поширили модель машини з довільним доступом до пам'яті (random access machine, RAM), яка є більш виразною за модель Шьонхаге.

Ганді у 1980 р. звернув увагу, на те, що машина Тюрінга — це машина послідовної дії і вона не відображає властивостей паралельних алгоритмів. Тобто машина Тюрінга та відповідні їй похідні моделі виконують однорівневі алгоритми — алгоритми з одним рівнем ієрархії. А паралельні алгоритми мають принаймні два рівні ієрархії: обчислення всередині процесорного вузла та обчислення, які розподілені серед багатьох таких вузлів. Тоді він став розробляти свою модель і висунув важливі принципи, яким вона повинна підлягати. Найважливішим є "принцип локальної причинності", який відкидає можливість миттєвого впливу операндів на відстані. Доведено, що якщо пристрій задовольняє цей принцип, тоді його послідовні стани утворюють послідовність коректних обчислень.

Юрій Гуревич у 1985 р. запропонував модель машини з абстрактними станами (abstract state machine — ASM). Основною відміною цієї моделі є те, що її стан зображується певною алгебраїчною структурою, тобто множиною конструктивних об'єктів з відповідного ансамбля та набором допустимих операцій з ними. Для цього було введене поняття алгебри, що розвивається (evolving algebra). Це дає змогу наочно задавати алгоритми, що змінюються динамічно. Іншою відміною є та, що ці конструктивні об'єкти можуть бути ієрархічними і таким чином, зображені алгоритми можуть бути будь-якої складності і форми.

Нарешті, у 1995 р. Гуревичем була запропонована паралельна ASM-модель для подання паралельних алгоритмів.

Обчислювальний процес у моделі ASM можна зобразити таким, як на Рис. 1.9. Наразі модель ASM успішно використовується для моделювання алгоритмів, заданих, наприклад, мовою C++.

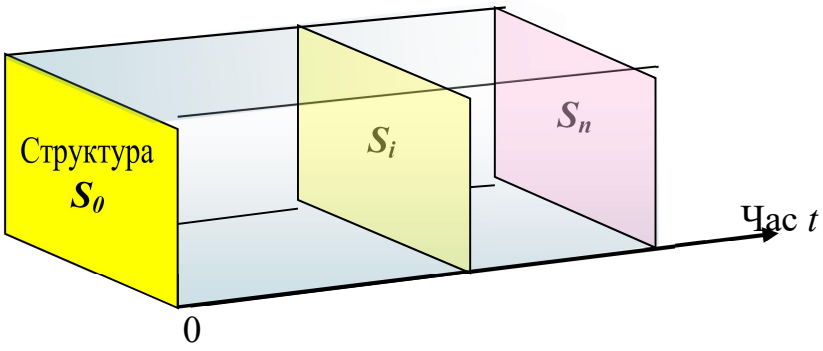


Рис. 1.9. Обчислювальний процес у моделі ASM

Отже, вище були розглянуті такі поняття, як конструктивний об’єкт, математична структура як модель обчислень, що складається з множини конструктивних об’єктів та системи відношень чи операторів, а також обчислювальний процес. Все це являє собою аспекти поняття алгоритму, яке існує як інтуїтивне поняття, а не як строго обмежене визначення. Таке визначення може бути сформульоване як наступне.

1.11.2 Графічне подання алгоритму

Кожна модель обчислювача передбачає формальне подання на ній алгоритму, яке є специфічним для неї. Для машини Тюрінга — це задана таблиця команд і переходів. Для алгорифмів Маркова — схеми підстановки. Для машин Колмогорова — множина команд та їх передування. Але у будь-якому випадку, такі подання можна зобразити у вигляді специфічного конструктивного об’єкта. Такий об’єкт вміщує в собі повну інформацію про алгоритм, яка закодована за певними правилами, характерними для даного виду алгоритмів.

Особливим видом подання алгоритму є ***графова модель***. Вершини $v_i \in V$ графу (V, E) алгоритму означають його операції, а ребра $e_j \in E$ — переходи між ними (Рис. 1.10).

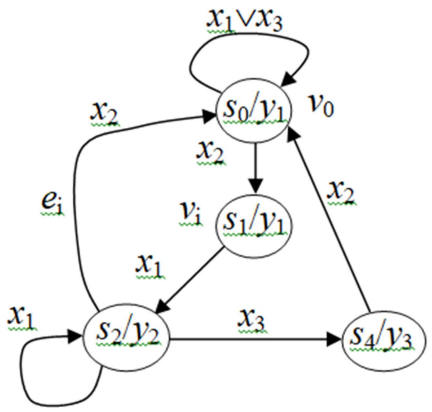


Рис. 1.10. Графова модель алгоритму

Виконання алгоритму полягає в послідовному обході вершин у напрямку ребер, починаючи з початкової вершини v_0, що задіює початкові дані, і закінчуючи кінцевою вершиною v_n, яка дає результат. У кожен момент часу одна вершина є активною, причому активна вершина v_0 генерує початковий стан $\boldsymbol{S}_0$, вершина v_k визначає стан $\boldsymbol{S}_i$. Семантика графа алгоритму пояснюється у підрозділі 4.3.

Аналогічні властивості має модель у вигляді ***блок-схеми алгоритму***. Це фактично граф алгоритму, вершини якого мають розширені теги, в яких явно задана операція алгоритму та характер вихідних ребер (в умовних вершинах), дуги мають теги з іменами відповідних змінних (Рис. 1.11).

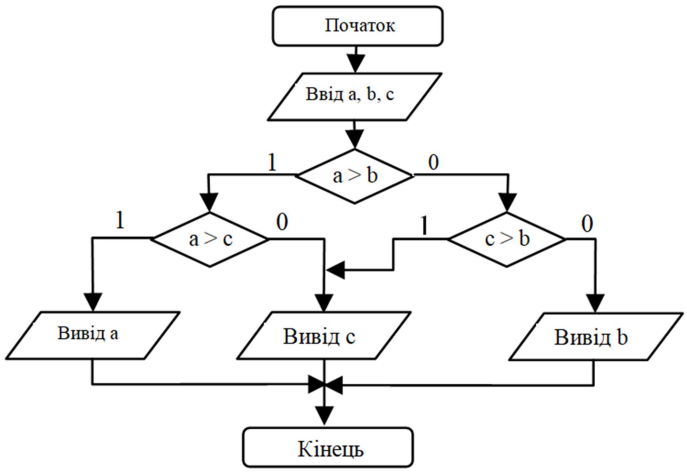


Рис. 1.11. Алгоритм у вигляді блок-схеми

1.11.3. Визначення алгоритму

Алгоритм — це обчислювальний процес обчислення функції в обчислювальній моделі, яка описується за допомогою строгих математичних понять [2].

Саме так можна читати визначення алгоритму, яке дане Тюрінгом. Поняття алгоритму передбачає,

по-перше, що повинен бути певний суб'єкт або процесор, який вміє читати, розпізнавати конструктивні об'єкти, операції та правильно виконувати ці операції з об'єктами відповідно до алгоритму.

По-друге, звичайно, алгоритм призначений для ефективного обчислення деякої корисної функції для отримання правильного результату для певного виду вхідних даних, тобто для ансамблю конструктивних об'єктів. Крім того, досягнення такого результату, який належить до ансамблю результатів, має бути гарантованим за скінченну кількість кроків.

По-третє, для полегшення розуміння та реалізації алгоритму, незначні деталі обчислювального процесу (які зазвичай не відображені в обчислювальній моделі) не враховуються.

На практиці, використовується набір різних обчислювальних моделей. Вони розрізняються як областю застосування, так і на рівні абстракції. В обох ситуаціях модель повинна бути доволі нескладною, щоб спрощувати організацію процесу обчислення, тобто, для складання алгоритму.

Концепція конструктивних об'єктів також слугує для того, щоб не розглядати незначні деталі алгоритму. Ансамблі конструктивних об'єктів зазвичай являють собою набір абстрактних рівнів. Наприклад, якщо ми маємо обробку багатьох масивів даних, то ми можемо не розглядати окремі дані цих масивів, тобто, абстрагуватись від них. Крім того, ми не маємо брати до уваги обробку окремих бітів цих даних.

Отже, концепція алгоритму нерозривно пов'язана з поняттям обчислювального процесу та обчислювальної моделі, які стосуються конструктивних об'єктів (див. Рис. 1.12). Тут до обчислювальної моделі, крім системи операцій та ансамблю конструктивних об’єктів, відноситься також пам’ять.

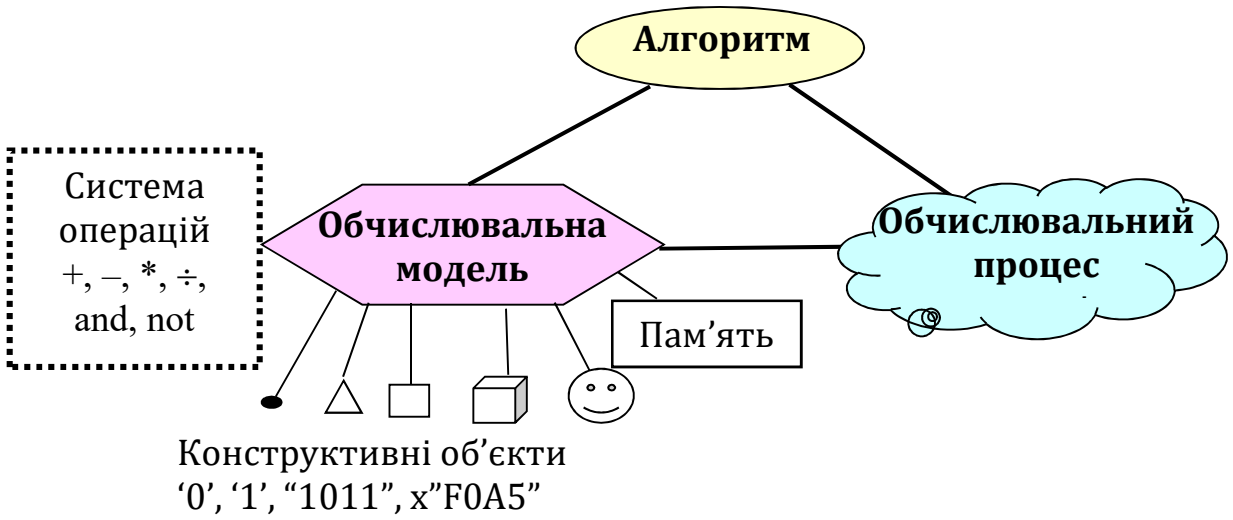


Рис. 1.12. Концепція алгоритму

Пам’ять — це місце, де тимчасово зберігаються конструктивні об’єкти, які можуть з неї вибиратись згідно з колмогоровським принципом локальності дії операцій. Власне, зміна станів пам’яті і є обчислювальним процесом. Пам’ять мають, явно чи неявно, усі обчислювальні моделі. Наприклад, стрічка машини Т’юрінга є такою пам’яттю нескінченного об’єму.

Коли ми говоримо про деякий обчислювальний процес, то ми вважаємо, що він відбувається в заданій обчислювальної моделі. Коли ми маємо належну обчислювальну модель, то ми можемо організувати в ній певний обчислювальний процес. І в обох цих ситуаціях ми маємо справу з алгоритмом, визначення якого залишається не до кінця точним.

Таким чином, алгоритм як подання обчислювального процесу

може існувати:

— окремо від обчислювальної моделі, як список правил чи як програма;

— як частина моделі — у ролі пристрою керування в машині Тюрінга;

— як сама модель, наприклад, модель скінченного автомату чи графова модель.

Наприклад, якщо виконавець алгоритму — це людина або програмований процесор, тоді алгоритм традиційно визначається як перелік інструкцій, які слід виконувати послідовно.

1.11.4. Загальні риси алгоритмів

Зараз будь-який алгоритм, що використовується на практиці, має відповідати наступному ряду вимог.

Дискретність. Стани обчислювального процесу змінюються у дискретні моменти часу, які позначені кроками чи тактами.

Детермінованість. Кожен наступний стан обчислювального процесу повністю визначається попереднім станом. Конкретному набору початкових об'єктів відповідає набір об'єктів-результатів.

Елементарність кроків. Кожен наступний стан одержується за елементарною операцією, яка виконується над попереднім станом з умовою локальності даних.

Спрямованість. Виконання алгоритму спрямоване на одержання очікуваного результату у визначеному кінцевому стані.

***Масовість*.** Початкові дані належать до ансамблю конструктивних об'єктів, потужність якого потенційно є нескінченною.

Для діяльності програміста обчислювальна модель набуває вигляду архітектури комп'ютера. Можна дати наступне визначення архітектури

та інших понять, пов'язаних із дискретною математикою, алгоритмами і структурами даних.

Архітектура — це модель реального комп'ютера з рівнем деталізації, що є достатнім для його розробки або програмування з забезпеченням реалізації відповідної множини алгоритмів. У більш широкому смислі, комп’ютерна архітектура — це набір правил і методів, які описують функціональні можливості, організацію та реалізацію комп’ютерних систем.

Програма є алгоритмом, який розробляється для певної архітектури комп'ютера за допомогою алгоритмічної мови або машинних кодів у залежності від рівня деталізації архітектури або рівня абстракції.

Отже, програма, яка написана для певної архітектури, повинна бути обчислена для тих самих вхідних даних з однаковими проміжними та кінцевими результатами на різних комп'ютерах, що мають однакову архітектуру.

Архітектура з точки зору розробника — це комп'ютерна модель з рівнем деталізації, яка є достатньою для його розробки та виробництва. Архітектура, як правило, включає в себе інформацію про набір команд, адресний простір, управління адресами, систему переривань, захист пам'яті, інтерфейси, периферійні пристрої, тобто всю інформацію про детальне технічне завдання для розробки комп'ютера. Це, так звана, архітектура рівня системи команд. Опис цієї архітектури не включає інформацію про елементну базу, її швидкодію, розміри комп'ютера, наявність та розміри кеш-пам’яті, споживання електроенергії, параметри безпеки тощо, оскільки вони безпосередньо не впливають на обчислювальний процес.

Архітектура з точки зору програміста — це комп'ютерна модель з рівнем деталізації, який є достатнім для успішного програмування певних обчислювальних задач на вибраній алгоритмічній мові.

Наприклад, програміст на мові Сі враховує модель пам'яті комп'ютера, до якої звертаються по 32-розрядній шині, з арифметико-логічним пристроєм (АЛП), який обробляє цілі числа або числа з плаваючою комою, з пам'яттю довільної ємності на жорсткому магнітному диску, клавіатуру та дисплей, доступ до яких забезпечується процедурами, які програміст може знайти в бібліотеках функцій мови Сі.

Кожний рядок алгоритму, який описаний мовою асемблера, означає конкретну машинну команду, що виконує елементарну операцію з даними або керує вибором наступної команди. Тому програмісту мовою асемблера потрібно добре знати архітектуру комп'ютера в деталях, до окремого регістра та його біта, коду переривання, адреси периферійного пристрою тощо.

Деякі архітектури для мови високого рівня визначаються як інтерфейс між мовою та системним програмним забезпеченням, яке безпосередньо реалізує скомпільовану програму. Таким інтерфейсом, як правило, стає операційна система або, так звана, віртуальна машина.

1.12. Структура дискретної математики

Отже, у даному розділі були послідовно розглянуті основні поняття і сутності, на яких ґрунтується дискретна математика. Структуру дискретної математики можна зобразити фігурою на Рис. 1.13. Вона відображає ієрархічність основних сутностей, які застосовуються у цій науці. На нижньому рівні розташовані конструктивні об’єкти, з яких будуються міркування. Наступний рівень складають множини конструктивних об’єктів. Між елементами множин задаються різноманітні відношення та функції. У свою чергу, на основі множин, відношень і функцій формуються математичні структури і алгебри.

Рис. 1.13. Структура дискретної математики

І нарешті, алгоритми представляються як процеси зміни у часі конструктивних об'єктів, множин, відношень і навіть структур та алгебр.

1.13. Завдання

1. Нехай $\mathcal{A}=\{1,2,3\}$ та $\mathcal{B}=\{a,b\}$. Визначте $\mathcal{A}\times\mathcal{B}$, $\mathcal{A}\times\mathcal{A}$, $\mathcal{B}\times\mathcal{A}$, $\mathcal{A}\times\varnothing$.

2. Нехай $A=\varnothing$, $\mathcal{B}=\{\varnothing\}$ та $C=\{\varnothing,\{\varnothing\}\}$. Визначте $A\times\mathcal{B}$, $A\times C$, $\mathcal{B}\times C$, $\mathcal{B}\times\mathcal{B}$, $C\times C$.

3. Перелічити елементи множини $A=\{x\in\mathbb{Z}\mid x^2-5x-15\le 0\}$

4. Визначте, які з властивостей (рефлексивність, симетричність, антисиметричність, транзитивність) мають наступні відношення на множині $\{1, 2, 3, 4, 5\}$:

R_1: $aR_1b \leftrightarrow |a-b|=1$;

R_2: $aR_2b \leftrightarrow 0<a-b<3$;

R_3: $aR_3b \leftrightarrow a+b$ — парне число;

R_4: $aR_4b \leftrightarrow a\ge b^2$;

R_5: $aR_5b \leftrightarrow$ НСД$(a,b)=1$.

5. Визначте які наступні твердження коректні:

1) будь-яке відношення на множині або симетричне, або антисиметричне;

2) жодне відношення не може бути одночасно симетричним та антисиметричним;

3) якщо відношення R є рефлексивним, симетричним, антисиметричним або транзитивним то і R^{-1} також має таку саму властивість.

6. Які з наступних відношень на множині цілих чисел $\mathbb{Z}$ є відношеннями порядку:

R_1: $aR_1b \leftrightarrow a \le b$;

R_2: $aR_2b \leftrightarrow a \ge b$;

R_3: $aR_3b \leftrightarrow a < b$;

R_4: $aR_4b \leftrightarrow a^2 \ge b^2$;

R_5: $aR_5b \leftrightarrow a = b$;

R_6: $aR_6b \leftrightarrow a$ ділиться на b ;

R_7: $aR_7b \leftrightarrow НЗД\ (a, b) = 1$?

7. Довести, що для довільних відношень R_1 і R_2 виконується

$(R_1 \circ R_2)^{-1} = R_2^{-1} \circ R_1^{-1}$.

8. Доведіть, що структура парних чисел з операцією «+» є моноїдом, а непарних чисел — ні.

9. Доведіть, що множина парних чисел за додаванням є абелевою групою.

10. Доведіть, що множина цілих чисел за множенням не є групою.

11. Доведіть, що група поворотів вектора на n різних кутів є абелевою групою.

12 Доведіть, що якщо кільце є комутативним, то аксіома $К_6$ повинна мати одне рівняння.

13. Поясніть, чому деякі графи називають ізоморфними.

14. Складіть алгоритм ділення цілих чисел з одержанням остачі, який має виконувати людина, що знає операції визначення знаку числа, додавання, віднімання, ділення пополам, вміє читати і записувати числа на папері.

2. Теорія множин

2.1. Множини

2.1.1. Визначення множини

Множина (англійською — set) — це будь-яка певна сукупність об'єктів. Об'єкти, з яких складена множина, називаються його ***елементами***. Елементи множини повинні відрізнятись один від одного як конструктивні об’єкти. У математиці множини та їх елементи мають імена або символьні позначення.

Множина, що не містить елементів, називається ***порожньою*** і позначається як $\emptyset$.

Множини як об'єкти можуть бути елементами інших множин. Множину, елементами якої є множини, іноді називають ***сімейством***.

Для виразного відображення ієрархії елементів та множин, елементи позначають малими буквами алфавіту, наприклад, *a, в, c,* множини позначають прописними буквами, як *A, B, C*, а сімейства — прописними буквами рукописного шрифту — $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$.

Зазвичай, у конкретних міркуваннях елементи множин, які розглядаються, беруться з певної однієї множини U з достатньою потужністю. Причому множина U формується в кожному випадку окремо. Така множина називається універсальною множиною або ***універсумом***.

Щоб задати множину, потрібно вказати, які елементи їй належать. Це може бути здійснено трьома способами:

— перерахуванням елементів: $M = \{a, b, c, ..., z\}$;

— за допомогою характеристичного предикату: $M = \{x \mid P(x)\}$;

— через породжувальну процедуру:

$$M = \{x_i \mid x_0 = 1, x_{i+1} = x_i + 1, i=0,...,9\}.$$

При поданні множин перерахуванням позначення елементів поділяють комами. Характеристичний предикат $P(x)$ — це деяка умова, яка виражена у формі логічного твердження, що дає змогу перевірити, чи належить даний елемент до множини. Якщо для даного елемента з універсуму U умова виконана, то він належить цій множині, в іншому випадку — не належить. Наприклад: $M = \{x \mid x \in \mathbb{N}, x < 10\}$.

Породжувальна процедура — це процедура, яка в процесі виконання породжує об'єкти, які є елементами множини, що визначається.

При поданні множин перерахуванням елементи іноді супроводжують індексами і вказують множину, з якої індекси беруться. Зокрема, запис $\{a_i \mid i = 1, \ldots, k\}$ означає те ж саме, що й $\{a_1, \ldots, a_k\}$, а запис $\mathcal{M} = \{M_a \mid a \in A\}$ означає, що $\mathcal{M}$ є сімейством, елементами якого є множини, причому індекс a «пробігає» значення з множини A. Знак трикрапки (...), який вживається при поданні множин, є скороченою формою запису, в якому породжувальна процедура для множини індексів вважається очевидною.

Кількість елементів множини M називається її ***потужністю*** (англійською — cardinality) і позначається як $|M|$, іноді як $\mathrm{card}(M)$. Згідно з потужністю, розрізняють скінченні та нескінченні множини.

Перерахуванням елементів можна задавати лише ***скінченні*** множини. Дуже великі чи нескінченні множини задаються характеристичним предикатом або породжувальною процедурою.

Можливість подання множин характеристичним предикатом залежить від предиката. Використання деяких предикатів для цієї мети може призводити до суперечностей. Наприклад, всі розглянуті в прикладах множини не містять себе як власного елемента. Розглянемо множину $\mathcal{B}$ таких множин, що не містять себе як елемент:

$$\mathcal{B} = \{A \mid A \notin A\}.$$

Якщо множина $\mathcal{B}$ існує, то ми повинні мати можливість відповісти на наступне питання: чи належить $\mathcal{B}$ до $\mathcal{B}$? Із визначення цієї множини випливає, що $A \in \mathcal{B}$, причому $A \notin A$. Отже, таке саме відношення стосується і самої множини $\mathcal{B}$, тобто, якщо $\mathcal{B} \in \mathcal{B}$, то $\mathcal{B} \notin \mathcal{B}$, і навпаки, із $\mathcal{B} \notin \mathcal{B}$ випливає, що $\mathcal{B} \in \mathcal{B}$. Тобто виникає непереборне логічне протиріччя, яке відоме як ***парадокс Рассела***.

Дуже схожий на парадокс Рассела парадокс цирюльника, представлений у наступній задачі. У одному селі жили селяни $a \in A$, які нездатні були себе голити. Там же жив селянин-цирюльник $x \in A$, який голив тільки таких селян. Питання: чи може цирюльник голити самого себе?

Рішення. Нездатність усіх селян голитись означає, що $\forall a \in A\ (\{a,a\} \notin P)$, де $P \subset A \cdot A$ – відношення спроможності голити. І ось знайшовся цирюльник x, спроможній голити усіх $\forall a \in A (\{a,x\} \in P)$. Слід довести, що $\{x,x\} \in P$. Але це довести неможливо, бо $\forall x \in A\ (\{x,x\} \notin P)$. Відповідь: такого цирюльника не існує.

Як бачимо, даний парадокс має рішення і його відміна від парадоксу Рассела в тому, що неможливо представити, що сам цирюльник належить до відношення спроможності голити, тобто $x \in P$. Отже, тут множини A і P складаються з об'єктів різної природи, як кажуть, з різних класів: $a \in A$ та $\{a,x\} \in P$. Але якщо розглядати абстрактні множини, то можливо і $x \in P$. Така належність цілком можлива при програмній реалізації множин та їх відношень.

У зв'язку з цим, можна розглянути такий приклад парадокса Расела.

Нехай A – це програмна процедура, яка є послідовністю (множиною) викликів процедур і відомо, що вона не може викликати сама себе, тобто, $A \notin A$ (A не є рекурсивною). B – це процедура яка складається з викликів процедур A, тобто, $B = \{A \mid A \notin A\}$.

Питання: чи існує така процедура B, яка викликає сама себе, тобто, $B \in B$?

Рішення. Нехай В викликає сама себе, тобто, $B \in B$, тоді оскільки $B = \{A\} = \{B\}$, а $A \notin A$, то і $B \notin B$. Виходить, з одного боку така процедура B – існує, а з іншого – не існує.

Протиріччю парадокса Рассела можна запобігти принаймні трьома способами:

1. Явно заборонити використання предикату належності множини собі самій $A \in A$.

2. Використовувати предикати на основі універсальної множини U, так що $P(x) = x \in U \wedge Q(x)$, де $Q(x)$ — довільний предикат. Тут умова належності цих елементів універсуму запобігає входженню множини у себе.

3. Призначити різні типи, класи множинам та їх елементам, так що предикати $P(x)$ виконуються лише з об'єктами відповідного типу і отже, стає неможливим виконання такого предикату, як $A \in A$. Така типізація є натуральною в сучасних мовах програмування і тому в програмістській практиці зараз парадокс Рассела, як правило, не відбувається.

2.1.2. Мультимножини

У множині всі елементи різні, а отже, кожен з них входить у множину рівно один раз. У деяких випадках виявляється корисним розглядати сукупності елементів, у які елементи входять по кілька разів. Наприклад, у перелік елементів схеми елементи одного типу входять кілька разів.

Нехай $X = \{x_1, ..., x_n\}$ — певна множина і нехай $a_1, ..., a_n \in \mathbb{N}$. Тоді ***мультимножиною*** (англійською — multiset, mset, bag) $\hat{X}$ над множиною X називається сукупність елементів множини X, у яку елемент x_i входить a_i разів. Мультимножина позначається одним із таких способів:

$$\hat{X} = \{x_1^{a_1}, \ldots, x_n^{a_n}\} = \{a_1(x_1), \ldots, a_n(x_n)\} = \{(x_1,a_1), \ldots, (x_n,a_n)\} = \{\underbrace{x_1, \ldots, x_1}_{a_1}; \ldots; \underbrace{x_n, \ldots, x_n}_{a_n}\}.$$

Тут X — це ***носій*** мультимножини, число a_i — ***показник*** елемента x_i, число $a_1 + \ldots + a_n$ — ***потужність*** мультимножини. Елементи мультимножини, так само, як і елементи множини, вважаються невпорядкованими.

Мультимножина $\hat{X}' = \{(x_1, a_1), \ldots, (x_n, a_n)\}$ називається ***індикатором***, якщо $\forall\, i \in 1, \ldots, n\ (a_i = 0 \vee a_i = 1)$, а множина $\underline{\hat{X}} = \{x_i \in X \mid a_i > 0\}$ називається складом мультимножини $\hat{X}$.

Приклад. Є множина $X = \{a, b, c\}$. Тоді для множини показників $A = \{2, 3, 0\}$ мультимножина виражається як

$$\hat{X} = \{a^2, b^3\} = \{2(a), 3(b)\} = \{(a, 2), (b, 3)\} = \{a, a, b, b, b\}.$$

Тут $\{a, b, c\}$ — індикатор мультимножини $\hat{X}$, а $\{a, b\}$ — її склад.

2.1.3. Операції над множинами. Порівняння

Операції над множинами — це способи порівняння множин та конструювання нових множин на основі наявних.

Перш за все, множини можна порівнювати між собою. Множина A ***міститься*** в множині B (множина B ***включає*** множину A), якщо кожен елемент множини A є елементом множини B, тобто:

$$A \subset B \stackrel{\text{def}}{=} \forall\, x \in \text{A}\ (x \in B).$$

Відношення між множинами та операції з ними зручно зображати за допомогою діаграм Венна. У такій діаграмі якусь множину зображають певною фігурою, наприклад, овалом. Таким чином, овал обмежує множину елементів, які можна зобразити певними точками на площині, кількість і розташування яких є довільними. Наприклад, те, що множина A міститься у множині B, зображується так, як на Рис. 2.1. Тут з діаграми

Венна є очевидним, що усі точки, які обмежені овалом A, входять у множину, яка обмежена овалом B.

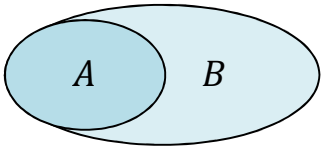


Рис. 2.1. Входження множини A у множину B

Універсальну множину зображають як прямокутник. Отже, вираз $A \subset U$ відображається як на Рис. 2.2.

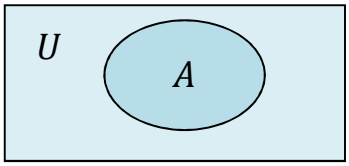


Рис. 2.2. Входження множини A в універсум

Дві множини дорівнюють одна одній, якщо вони є підмножинами одна одної:

$$A = B \stackrel{\text{def}}{=} A \subset B \wedge B \subset A.$$

Якщо $A \subset B$ і $A \neq B$, то множина A називається ***власною підмножиною*** множини B, а B — ***власною надмножиною*** множини A. Якщо підмножина A є ***невласною***, то таке відношення позначається як $A \subseteq B$, тобто A міститься в B чи дорівнює B.

Включення множин має такі властивості, які розглядаються як теореми:

T1: $\forall A\ (A \subset A)$.

T2: $\forall A, B\ (A \subset B \wedge B \subset A \Rightarrow A = B)$.

T3: $\forall A, B, C\ (A \subset B \wedge B \subset C \Rightarrow A \subset C)$.

Теорема T1 випливає з визначення включення: $\forall x \in A\ (x \in A)$. Теорема T2 є, власне, визначенням. Теорему T3 можна наочно довести за допомогою діаграми Венна на Рис. 2.3.

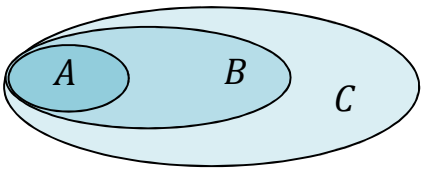


Рис. 2.3. Доведення теореми T_3

Між множинами A і B встановлена ***взаємно однозначна відповідність***, якщо кожному елементу множини A поставлений у відповідність один і тільки один елемент множини B і для кожного елемента множини B поставлений у відповідність один і тільки один елемент множини A. У цьому випадку говорять також, що множини A і B є ***ізоморфними***, що позначається як $A \sim B$.

Якщо при заданій відповідності елементу $a \in A$ відповідає елемент $b \in B$, то дану обставину позначають наступним чином: $a \longmapsto b$. Наприклад, операція $n \longmapsto 2n$ встановлює взаємно однозначну відповідність між множиною натуральних чисел $\mathbb{N}$ і множиною парних натуральних чисел $2\mathbb{N}$, тобто $\mathbb{N} \sim 2\mathbb{N}$.

Якщо між двома множинами A і B може бути встановлена взаємно однозначна відповідність, то кажуть, що множини мають однакову потужність, або що множини ***рівнопотужні***, і записують це так: $|A| = |B|$. Це записується як

$$|A| = |B| \stackrel{\text{def}}{=} A \sim B.$$

Згідно з цим визначенням, множина пальців на руках людини рівнопотужна множині десяткових цифр. Також множина $\mathbb{N}$ рівнопотужна множині парних чисел $2\mathbb{N}$.

У останньому випадку у непідготовленої людини закрадаються сумніви. З античних часів вважається за аксіому принцип: частина є меншою за ціле. Тоді якщо $\mathbb{N}$ — нескінченна множина, то виходить, що множина $2\mathbb{N}$ — ще більш нескінченна? А з іншого боку, оскільки множина $2\mathbb{N}$ формується з множини $\mathbb{N}$ відкиданням кожного другого елементу, тобто, вона менша за $\mathbb{N}$?

Але виявляється, що цей принцип годиться лише для скінченних множин. Ясно, що множина парних чисел є підмножиною осяжної множини цілих чисел. Проте, якщо між двома нескінченними множинами встановлена взаємно однозначна відповідність, то такі множини є рівнопотужними за визначенням. Щоб закріпити поняття нескінченності множин, введено наступне визначення.

Множина A є ***скінченною***, якщо у неї немає рівнопотужної власної підмножини:

$$|A| < \infty \stackrel{\text{def}}{=} \forall B\,((B \subset A \wedge |A| = |B|) \Rightarrow A = B).$$

Решта множин вважаються нескінченними. Отже, для ***нескінченної*** множини одержимо через заперечення умови скінченності:

$$(|A| = \infty) \stackrel{\text{def}}{=} \exists B\,(B \subset A \wedge |A| = |B| \wedge A \neq B).$$

Іншими словами, нескінченна множина є рівнопотужною певній своїй підмножині. Звідси, множина $\mathbb{N}$ є нескінченною, бо є рівнопотужною своїй підмножині парних чисел $2\mathbb{N}$.

При дослідженні множини натуральних чисел та порівнянні її з іншими нескінченними множинами її потужність позначають івритською літерою алеф: $|\mathbb{N}| = \aleph_0$ (читається як алеф нульовий).

Для порівняння, потужність множини реальних чисел дорівнює потужності континууму і позначається як $|\mathbb{R}| = \mathfrak{c}$. Відповідно, $\aleph_0 < \mathfrak{c}$. Існує гіпотеза Кантора, яка стверджує, що $\mathfrak{c} = 2^{\aleph_0}$. Її наслідком є доведення того, що між цілими та реальними числами не існують якісь проміжні числа (згадаємо, що раціональні числа виражаються через цілі). Тобто, в ряду

гіпотетичних нескінченних потужностей $\aleph_0 < \aleph_1 < \aleph_2 \ldots$ наступний за $\aleph_0$ є елемент $\aleph_1 = 2^{\aleph_0}$.

Для дослідження скінченних множин є важливою наступна теорема.

$$\text{T: } \forall A\ (A \neq \emptyset \wedge |A| < \infty \Rightarrow \exists\, k \in \mathbb{N}\ (|A| = |\{1, \ldots, k\}|)).$$

Вона означає, що будь-яка скінченна непуста множина рівнопотужна певному відрізку натурального ряду. Таким чином, якщо множина A скінченна, $|A| = k$, то елементи A завжди можна перенумерувати, тобто поставити їм у відповідність номери з відрізку $1, \ldots, k$ за допомогою деякої процедури. Власне, доведення цієї теореми виконується через таку процедуру. Наявність такої процедури мається на увазі, коли застосовується подання множини, за якого її елементи мають цілочисельні індекси, як наприклад, $X = \{x_1, \ldots, x_n\}$.

У обчислювальній техніці дуже поширена нумерація елементів множин цілими числами. Так, нумеруються такі елементи множин, як розряди двійкових чисел, комірки оперативної пам'яті, елементи масивів. Таким чином, адресація програмних змінних — це доступ до них за відповідним номером, який зображується цілим числом.

2.1.4. Основні операції над множинами

Над множинами визначені наступні операції:

об'єднання:

$A \cup B \stackrel{\text{def}}{=} \{x \mid x \in A \vee x \in B\}$;

перетин:

$A \cap B \stackrel{\text{def}}{=} \{x \mid x \in A \wedge x \in B\}$;

різниця:

$A \setminus B \stackrel{\text{def}}{=} \{x \mid x \in A \wedge x \notin B\}$;

симетрична різниця:

$A \dot{-} B \stackrel{\text{def}}{=} (A \cup B) \setminus (A \cap B) = \{x \mid (x \in A \wedge x \notin B) \vee (x \notin A \wedge x \in B)\}$;

доповнення:

$\bar{A} \stackrel{\text{def}}{=} \{x \mid x \notin A\}$;

Операція доповнення має на увазі, що існує деякий універсум U, так що $\bar{A} = U \setminus A$. В іншому випадку операція доповнення не визначена. Операція доповнення позначається горизонтальною рискою над літерою або цілим виразом.

На Рис. 2.4 показані діаграми Венна для визначених вище операцій над множинами. Результати операцій виділені кольором.

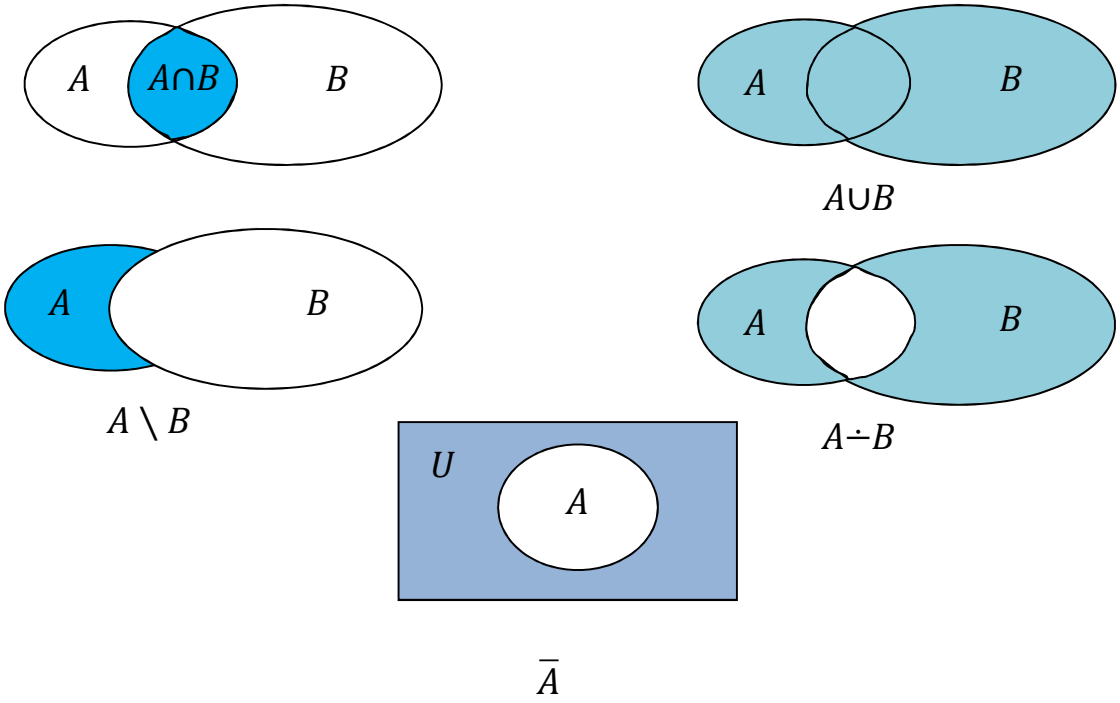


Рис. 2.4. Основні операції над множинами

Нехай $A = \{1, 2, 3\}$, $B = \{3, 4, 5\}$. Тоді

$A \cup B = \{1, 2, 3, 4, 5\}$, $A \cap B = \{3\}$, $A \setminus B = \{1, 2\}$, $A \dot{-} B = \{1, 2, 4, 5\}$.

Визначимо універсум як

$$U = \{0, 1, 2, 3, 4, 5, 6\},$$

тоді

$$\bar{A} = \{0, 4, 5, 6\}, \bar{B} = \{0, 1, 2, 6\}.$$

Якщо множини A і B скінченні, то з визначень операцій та діаграм на Рис. 2.4 можна визначити наступне.

$$|A \cup B| = |A| + |B| - |A \cap B|;$$

$$|A \setminus B| = |A| - |A \cap B|;$$

$$|A \div B| = |A| + |B| - 2|A \cap B|.$$

Операції перетину й об'єднання двох множин допускають узагальнення, коли та чи інша операція виконується для n множин.

Розбиття і покриття

Нехай $\mathcal{E} = \{E_i \mid i = 1, \ldots, n\}$ — деяке сімейство множин. Сімейство $\mathcal{E}$ називається ***покриттям*** множини M (Рис. 2.5), якщо кожен елемент $x \in M$ належить хоча б одному з E_i:

$$\forall x \in M \, (\exists i \, (x \in E_i)).$$

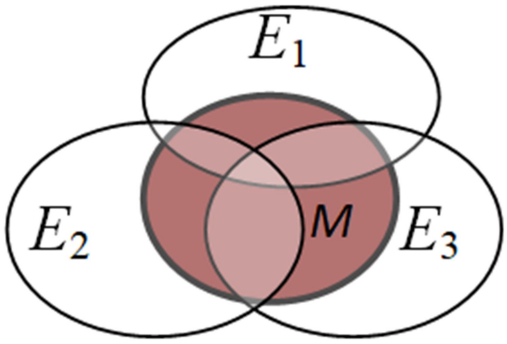


Рис. 2.5. Покриття $\{E_1, E_2, E_3\}$ для множини M

Звідси випливає, що об’єднання елементів покриття містить саму множину, тобто

$$M \subseteq \bigcup_i E_i.$$

Сімейство $\mathcal{E}$ називається ***диз'юнктним***, якщо елементи цього сімейства попарно не перетинаються (Рис. 2.6), тобто кожен елемент множини M належить не більш ніж одній з множин E_i, тобто,

$$\forall i, j \, (i \neq j \Rightarrow E_i \cap E_j = \emptyset).$$

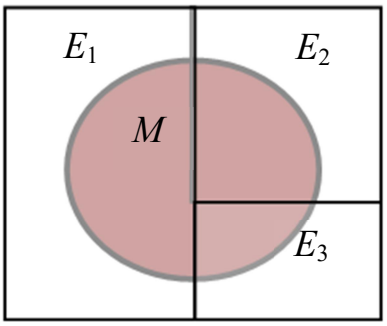


Рис. 2.6. Диз'юнктне покриття $\{E_1, E_2, E_3\}$ для множини M

Диз'юнктне покриття називається ***подрібненням*** множини M. Елементи розбиття, тобто підмножини множини M, часто називають ***блоками подрібнення.***

Якщо кожен з блоків подрібнення повністю належить множині M, тобто, $\forall i(E_i \subset M)$, то таке подрібнення називається ***розбиттям***.

Приклад. Нехай $M = \{1, 2, 3\}$. Тоді сімейство $\{\{1, 2\}, \{2, 3\}, \{4, 1\}\}$ є покриттям, але не подрібненням; сімейство $\{\{1\}, \{2\}, \{3, 4\}\}$ є і покриттям, і подрібненням, сімейство $\{\{1\}, \{2\}\}$ є диз'юнктним, але не є ні покриттям, ані подрібненням. А ось сімейство $\{\{1\}, \{2, 3\}\}$ є розбиттям.

Властивості операцій над множинами

Операції над множинами мають цілу низку важливих властивостей. Нехай визначений універсум U. Тоді для множин $\forall\ A,\ B, C \subset U$ виконуються наступні рівності:

1) ідемпотентність:

$A \cup A = A, \qquad A \cap A = A;$

2) комутативність:

$A \cup B = B \cup A, \qquad A \cap B = B \cap A, \qquad A \div B = B \div A;$

3) асоціативність:

$A \cup (B \cup C) = (A \cup B) \cup C$, $A \cap (B \cap C) = (A \cap B) \cap C$,
$A \dot{-} (B \dot{-} C) = (A \dot{-} B) \dot{-} C$;

4) дистрибутивність:

$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$, $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$,
$A \cap (B \dot{-} C) = A \cap B \dot{-} A \cap C$;

5) поглинання:

$(A \cap B) \cup A = A$, $(A \cup B) \cap A = A$;

6) властивості нуля:

$A \cup \emptyset = A$, $A \cap \emptyset = \emptyset$, $A \dot{-} \emptyset = A$;

7) властивості одиниці:

$A \cup U = U$, $A \cap U = A$;

8) інволютивність (involutiveness):

$\bar{\bar{A}} = A$;

9) закони де Моргана

$\overline{A \cap B} = \bar{A} \cup \bar{B}$, $\overline{A \cup B} = \bar{A} \cap \bar{B}$;

10) властивості доповнення (complement):

$A \cup \bar{A} = U$, $A \cap \bar{A} = \emptyset$;

11) закони склеювання (gluing)

$A \cap B \cup A \cap \bar{B} = A$, $A \cup \bar{A} \cap B = A \cup B$;

12) вирази для різниці:

$A \setminus B = A \cap \bar{B}$.

13) вирази для симетричної різниці

$A \dot{-} B = A \cap \overline{B} \cup \overline{A} \cap B$, $\overline{A \dot{-} B} = A \cap B \cup \overline{A} \cap \overline{B}$;

14) умова рівності

$A = B$, якщо і тільки якщо $A \div B = \varnothing$;

15) умова входження

$A \subset B$, якщо і тільки якщо $A \cap B = A$ або $A \cup B = B$, або $A \cap \overline{B} = \varnothing$.

У справедливості перелічених рівностей можна переконатися різними способами. Наприклад, можна намалювати діаграми Венна для лівої і правої частин рівності й переконатися, що вони збігаються, або ж провести формальне міркування для кожної рівності.

Розглянемо для прикладу першу рівність: $A \cup A = A$. Візьмемо довільний елемент x, що належить лівій частині рівності, $x \in A \cup A$. За визначенням операції об'єднання, маємо $x \in A \vee x \in A$. У будь-якому випадку, $x \in A$. Взявши довільний елемент з множини в лівій частині рівності, виявимо, що він належить множині в правій частині. Звідси, за визначенням включення множин, отримуємо, що $A \cup A = A$. Нехай тепер $x \in A$ у правій частині. Тоді, очевидно, правильно $x \in A \vee x \in A$. Звідси, за визначенням операції об'єднання, маємо $x \in A \cup A$. Таким чином, $A \cup A = A$. Це міркування можна записати коротше:

$$x \in A \cup A \Leftrightarrow x \in A \vee x \in A \Leftrightarrow x \in A.$$

Аналогічні міркування неважко провести і для інших рівностей.

2.1.5. Задачі з множинами

Наведені співвідношення лежать в основі розв’язання задач з множинами, таких як спрощення виразу та розв’язання рівняння з одним невідомим.

Приклад. Спростити вираз $\overline{\overline{A} \cup B \cup (A \cup \overline{B})}$.

Застосувавши закон де Моргана (п. 9):

$$\overline{\overline{\overline{A} \cup B} \cup (A \cup \overline{B})} = \overline{\overline{\overline{A} \cup B}} \cdot \overline{A \cup \overline{B}} = (\overline{A} \cup B) \cdot \overline{A} \cdot \overline{\overline{B}} = (\overline{A} \cup B) \cdot \overline{A} \cdot B =$$

$$= \overline{A} \cdot \overline{A} \cdot B \cup B \cdot \overline{A} \cdot B = \overline{A} \cdot B = B \setminus A.$$

Розв’язування рівнянь і систем рівнянь з одним невідомим X в алгебрі множин можна виконати у три етапи.

1. Згідно з умовою, рівності лівої і правої частин рівняння (п.14), початкове рівняння приводиться до вигляду, що містить порожню множину в правій частині.

2. Отримане рівняння перетворюється до вигляду

$$M \cap X \cup N \cap \overline{X} = \varnothing,$$

де M і N — деякі множини або вирази в алгебрі множин, що не містять X. Нескладно показати, що будь-яке рівняння з порожньою правою частиною приводиться до вказаного вигляду.

3. На основі отриманого рівняння та умови входження (п.15) записується розв’язок у вигляді $N \subset X \subset \overline{M}$.

Приклад. Розв'язати рівняння $X \cup C = D$, $C \subseteq D$.

Використовуючи умови рівності (14), отримуємо $(X \cup C) \div D = \varnothing$.

Треба правильно застосовувати дужки при перетворенні виразів. Так, запис $X \cup C \div D = \varnothing$, який часто задіяний при цьому, допускає неправильне тлумачення, наприклад, $X \cup (C \div D) = \varnothing$, що призводить до помилок у подальшому розв'язуванні. Слід користуватися лише записом $(X \cup C) \div D = \varnothing$, тому що операція об’єднання вважається більш пріоритетною за операцію симетричної різниці.

Використовуючи вирази для симетричної різниці (п.13), прибираємо в отриманому рівнянні знак «÷». Маємо:

$$(X \cup C) \div D = \overline{X \cup C} \cdot D \cup (X \cup C)\overline{D} = \varnothing.$$

Далі застосовуємо теорему де Моргана і розкриваємо дужки. Отримуємо:

$$\overline{X}\ \overline{C}\ D \cup X\overline{D} \cup C\overline{D} = \varnothing.$$

Використовуючи умову входження $C \subset D$, отримуємо $C\overline{D} = \varnothing$.

Таким чином, рівняння приводиться до вигляду $\overline{X}\ \overline{C}\ D \cup X\overline{D} = \varnothing$. Звідси розв'язок може бути записаний так: $\overline{C}\ D \subset X \subset D$.

Приклад. Розв'язати рівняння $X \setminus A = C \setminus X$;

$(X \setminus A) \div (C \setminus X) = \varnothing$;

$(X \cap \overline{A}\ ) \div (C \cap \overline{X}\ ) = \varnothing$;

$(X \cap \overline{A}\ ) \cap \overline{C \cap \overline{X}} \cup \overline{X \cap \overline{A}} \cap (C \cap \overline{X}\ ) = \varnothing$;

$(X \cap \overline{A}\ ) \cap (\overline{C} \cup X)) \cup ((\overline{X} \cup A) \cap C \cap \overline{X}\ ) = \varnothing$;

$(X \cap \overline{A} \cap \overline{C} \cup X \cap \overline{A}\ ) \cup (C \cap \overline{X} \cup A \cap C \cap \overline{X}\ ) = \varnothing$;

$(\overline{A} \cap X) \cup (C \cap \overline{X}\ ) = \varnothing$;

Відповідь: $C \subset X \subset A$; ілюструється Рис. 2.7.

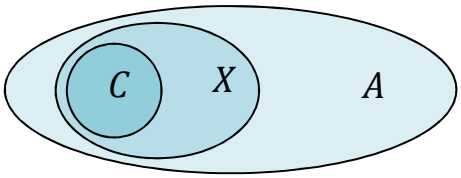


Рис. 2.7. Пояснення розв'язку

Приклад. Розв'язати рівняння $\overline{\overline{X \setminus A} \div B} = X$ за умови $A \cdot B = \varnothing$;

$\overline{\overline{X \setminus A} \div B} \div X = \varnothing$.

Використаємо, відповідно, вирази для різниці, симетричної різниці та інволютивності:

$$((\overline{\overline{X \setminus A}}\ \overline{B}) \cup \overline{X \setminus A} \cdot B) \div X = \varnothing;$$

$$[(X \setminus A)\overline{B} \cup \overline{X \setminus A}\ B] \div X = \varnothing;$$

$$(X\overline{A}\ \overline{B} \cup \overline{X\overline{A}}\ B) \div X = \varnothing.$$

Застосовуючи теорему де Моргана, отримуємо:

$$[X\overline{A}\ \overline{B} \cup (\overline{X} \cup A)B] \div X = \varnothing.$$

Розкриваємо дужки:

$$(X\overline{A}\ \overline{B} \cup \overline{X}\,B \cup AB) \div X = \varnothing.$$

Оскільки в умові рівняння задано, що $AB = \varnothing$, то

$$(X\overline{A}\ \overline{B} \cup \overline{X}\,B) \div X = \varnothing.$$

Використовуємо співвідношення для симетричної різниці:

$$(\overline{X\overline{A}\ \overline{B} \cup \overline{X}\,B})X \cup (X\overline{A}\ \overline{B} \cup \overline{X}\,B)\,\overline{X} = \varnothing.$$

Застосовуємо теорему де Моргана і розкриваємо дужки:

$$\overline{X\overline{A}\ \overline{B}} \cdot \overline{\overline{X}\,B} \cdot X \cup \overline{X}\,B = \varnothing.$$

Використаємо теорему де Моргана:

$$(\overline{X} \cup A \cup B)(X \cup \overline{B})\,X \cup \overline{X}\,B = \varnothing.$$

Розкриваємо дужки:

$$(\overline{X}\ \overline{B} \cup AX \cup A\overline{B} \cup BX)\,X \cup \overline{X}\,B = \varnothing.$$

Розкриваємо дужки:

$$AX \cup A\overline{B}\,X \cup BX \cup \overline{X}\,B = \varnothing$$

Приводимо рівняння до вигляду $MX \cup N\overline{X} = \varnothing$, групуючи члени:

$$X(A \cup A\overline{B} \cup B) \cup \overline{X}\,B = \varnothing.$$

Використаємо, відповідно, співвідношення поглинання та ідемпотентності:

$X(A \cup B \cup B) \cup \overline{X}\, B = \varnothing$; $X(A \cup B) \cup \overline{X}\, B = \varnothing$.

Розв’язок записується у вигляді

$B \subset X \subset \overline{A \cup B} = \overline{A} \cdot \overline{B}$.

2.1.6. Подання множин в програмах

Зобразити в програмі будь-який об'єкт (у даному випадку, множину) — це означає описати в термінах системи програмування структуру даних, яка використовується для зберігання інформації про цей об'єкт, а також алгоритми над такою структурою даних, які реалізують властиві даному об'єкту операції. Це вимоги до, так званої у програмуванні, ***абстрактної структури даних***. Отже, зображення множини має на меті опис способу зберігання інформації про належність елементів множині та опис алгоритмів для обчислення об'єднання, перетину й інших введених операцій. Слід відзначити, що, як правило, один і той самий об'єкт може бути зображений багатьма різними способами, причому не можна вказати спосіб, який є найкращим для всіх можливих випадків. Далі, як один з таких способів, розглядається бітова шкала, що заснована на понятті булеана.

Булеан

Сімейство всіх підмножин множини M називається ***булеаном*** множини M і позначається як 2^M:

$$2^M \stackrel{\text{def}}{=} \{A \mid A \subset M\}$$

Приклад. Є множина $M = \{1, 2, 3\}$. Її булеан:

$$2^M = \{\varnothing,\ \{3\},\ \{2\},\ \{2,3\},\ \{1\}, \{1, 3\}, \{1, 2\}, \{1, 2, 3\}\}.$$

Потужність булеана дорівнює $|2^A| = 2^n = 2^{|A|}$. Англійською булеан називається powerset, що наголошує на те, що це — певного роду множина в степені.

Бітові шкали

Нехай заданий скінченний універсум U і число елементів в ньому не перевищує розрядності комп'ютера, $|U| \le n$. Найчастіше $n = 32$ чи 64. Елементи універсуму нумеруються:

$$U = \{u_1, ..., u_n).$$

Підмножина A універсуму U зображується кодом (машинним словом або бітовою шкалою), яку для простоти розглянемо як бітовий масив C = (c[1], c[2],..., c[n]), у якому:

$$c[i] = \begin{cases} 1, \text{ якщо } u_i \in A, \\ 0, \text{ якщо } u_i \notin A. \end{cases}$$

Тобто i-та одиниця свідчить про те, що в дану множину входить i-й елемент універсуму. Нехай маємо універсум $U = \{a, b, c, d\}$. Тоді можливі множини для нього і відповідне зображення, які перелічені у Таблиці 2.1.

Таблиця 2.1 Кодування множин бітовою шкалою

№	Код	Множина	№	Код	Множина
0	0000	∅	8	1000	*a*
1	0001	*d*	9	1001	*a, d*
2	0010	*c*	10	1010	*a, c*
3	0011	*c, d*	11	1011	*a, c, d*
4	0100	*b*	12	1100	*a, b*
5	0101	*b, d*	13	1101	*a, b, d*
6	0110	*b, c*	14	1110	*a, b, c*
7	0111	*b, c, d*	15	1111	*a, b, c, d*

Отже, n-бітовий код кодує одну з 2^n можливих множин, які всі разом формують булеан 2^A множини A. Саме ця властивість є причиною етимології назви «булеан», тобто це множина відношень від n булевських чи двійкових елементів.

Якщо множина зображується машинним словом, то операції з такими множинами виконуються за допомогою логічних операцій системи команд процесора. Наприклад, доповнення множини $\bar{A}$ одержується виконанням операції інверсії двійкових розрядів слова.

Приклад. Є універсум $U = \{a, b, c, d\}$. Множина $A = \{a, b\}$, за Табл. 2.1, кодується кодом 1100, його інверсія дорівнює 0011, а отже, $\bar{A} = \{c, d\}$.

Нехай є множина $B = \{a, d\}$, її код 1001, код $A \cup B$ — 1100 ∨ 1001 = = 1101 і результат — $A \cup B = \{a, b, d\}$.

Аналогічно знаходиться перетин множин за допомогою операції порозрядного «І» та симетрична різниця — за допомогою операції порозрядне «Виключне Або».

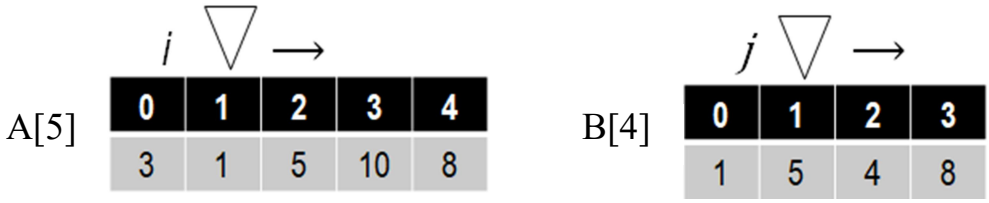
Якщо потужність універсуму перевершує розмір машинного слова (це, як правило, 32 або 64), але не дуже велика, то для подання множин використовуються масиви бітових шкал.

Якщо універсум дуже великий, а розглядаються його невеликі підмножини, то зображення за допомогою бітових шкал не є ефективним з точки зору економії пам'яті. Тоді універсум виражається як послідовність цілих чисел від 0 до n, а множину виражають як масив цілих чисел розміром n. Для того, щоби зменшити об’єм пам’яті, що займає така множина при великому n, множину формують як список завдовжки з потужність даної множини.

У файловій системі NTFS, яка застосовується у багатьох операційних системах, множина зайнятих кластерів диску, потужність якої досягає $> 10^8$, зображається саме бітовою шкалою. Таке подання є компромісом

між об'ємом даних про множину та швидкістю пошуку незайнятих кластерів, в які можна записувати файл.

Операції над множинами, які задані у вигляді масивів, виконують послідовно, скануючи масиви, що обробляються, у гнізді циклів: у внутрішньому циклі виконується обхід елементів одного масиву-множини, а у зовнішньому — другого масиву (Рис. 2.8). Отже, для виконання певної операції над множинами потрібно n^2 операцій. Це число скорочується при обробці масивів у вигляді списків. Але чергова операція сканування списку значно повільніша за операцію доступу до масиву.




```
k = 0; // A ∩ B
for (i = 0; i < 5; i++){
     for (j = 0; j < 4; j++){
          if (A[i] == B[j])
               C[k++] = A[i];
     }
}
```

Рис.2.8. Обчислення перетину множин A і B

Для того, щоби спростити та прискорити виконання операцій над множинами, ці множини зображають у вигляді хешованих таблиць. При цьому доступ до елемента множини виконується за його іменем, а не за номером чи адресою в масиві. Тоді операція над множинами виконується за порядку log$n$ команд процесора.

Такі структури, як масиви, списки та хешовані таблиці розглядаються у курсі, який вивчає структури даних.

2.1.7. Завдання

1. Які з наведених нижче співвідношень неправильні й чому?

1) $1 \in \{\{1, \{2, \{3, 4\}\}\}\}$;

2) $\{3, 4\} \in \{1, \{2, \{3, 4\}\}\}$;

3) $\{1, \{2, 3\}\} \in \{1, \{\{2, 3\}, 4\}\}$;

4) $\{1\} \in \{1, 2, \{\{1\}, 4\}\}$;

5) $\{1, 2\} \in \{1, 2, 3, 4\}$;

6) $\{1, 2\} \in \{1, 2\}$;

7) $1 \in \{\{1\}, \{2, \{3, 4\}\}\}$;

8) $\{1, \{2, 3\}\} \in \{1, \{\{2, 3\}\}\}$.

2. Чи пов'язані множини A і B відношенням включення? Якщо так, то вкажіть, котра з них є підмножиною іншої.

1) $A = \{1, \{2, 3\}\}$, $B = \{1, 2, 3\}$;

2) $A = \{1, 2\}$, $B = \{3, 4, \{1, 2\}\}$;

3) $A = \{1, \{2, \{3\}\}\}$, $B = \{1, 4, \{2, \{3\}\}\}$;

4) $A = \{1, \{2, \{3, 4, \{5, 6\}\}\}\}$, $B = \{2, \{3, 4, \{5, 6\}\}\}$;

5) $A = \{1, 2\}$, $B = \{1, 2, 3, 4\}$.

3. Визначити $A \cup B$, $A \cap B$, $A \setminus B$, $B \setminus A$, $A \div B$.

Множини брати з задачі 2.2.

4. Спростити вирази:

1) $\overline{\overline{A \cup \overline{A}\, B} \div A}$;

2) $A\,\overline{B} \div \overline{A}\, B \div \overline{A}\ \overline{B}$;

3) $\overline{(\overline{A \setminus B}) \setminus (\overline{A} \setminus \overline{B})}$;

4) $\left[\overline{AB \cup \overline{\overline{A} \cup \overline{B}}} \setminus (\overline{A} \cup \overline{B})\right] \div A$;

5) $\overline{\overline{A \div B} \div \overline{A}}$;

6) $\overline{\overline{\overline{AX} \cdot B \cdot \overline{B}} \cdot A}$;

7) $\overline{\overline{\overline{A \cup B} \cdot \overline{B}} \cup \overline{BA}}$.

2.1.5. Розв’язати рівняння:

1) $X \setminus A = C \setminus X$;

2) $\overline{(\overline{A} \cup \overline{X}) \cdot \overline{A\,\overline{X}}} \setminus X = B, B \subseteq A$;

3) $A \div C\overline{X} = C \setminus A, A \subseteq C$;

4) $\overline{(\overline{X} \cup A \cup \overline{C})\ \overline{X\overline{A}\ \overline{C}}} = C, A \cap C = \varnothing$;

5) $\overline{A} \div X \div B \div \overline{B} = B, B \subseteq A$;

6) $\overline{\overline{AX} \cdot \overline{AX \cdot \overline{AX}}} = C, C \subseteq A$;

7) $A \setminus (X \setminus C) = A \cup C$;

8) $(A \setminus X) \setminus C = A \cup C$;

9) $A \cap X = B$;

10) $A \cup X = B \cap X$;

11) $A \div X = B \cap X$;

12) $A \div X = B \cap X \setminus A$;

13) $A \cap \overline{X \cap B} = (X \setminus A) \cap B$;

14) $\overline{X \cup B} = \overline{X \cap B}$;

2.2. Відношення

2.2.1. Визначення

Вище було розглянуте відношення як одна з основ дискретної математики. Далі особливості відношення розглядаються детальніше. Перш за все, відношення визначаються на впорядкованих множинах. Найменша з них — ***впорядкована пара*** (a, b) з об'єктів a, b. Впорядкованість означає, що $(a, b) \neq (b, a)$. Формально така пара визначається через множину як $\{a, \{a, b\}\}$, тобто елемент a вважається першим.

Аналогічно до впорядкованих пар, можна розглядати впорядковані трійки, четвірки і т. д. У загальному випадку, подібні об'єкти називаються ***n-ками***, ***кортежами***, ***наборами*** або ***скінченними послідовностями***. Упорядкований набір з n елементів позначається як $(a_1,...,a_n)$.

Найбільш природним поданням в програмі кортежу з елементами множини A є масив змінних типу A довжиною n.

Прямий добуток множин

Нехай A і B — дві множини. ***Прямим*** (***декартовим***) ***добутком*** двох множин A і B називається множина всіх упорядкованих пар, в яких перший елемент належить A, а другий належить B:

$$A \times B \stackrel{\text{def}}{=} \{(a, b) \mid a \in A \wedge b \in B\}.$$

Наприклад, точка на площині може бути задана впорядкованою парою координат (x, y), тобто двома точками на координатних осях. Таким чином, такі пари належать раціональному полю $\mathbb{R}^2 = \mathbb{R} \times \mathbb{R}$. Тому своєю появою метод координат завдячує Декарту, звідси і назва «декартів добуток».

Поняття прямого добутку множин допускає узагальнення. Прямий добуток множин $A_1, ..., A_n$ — це множина наборів (кортежів):

$$A_1 \times ,..., \times A_n = \{(a_1,..., a_n) \mid a_1 \in A_1 \wedge ... \wedge a_n \in A_n\}.$$

У такому добутку множини A_i не обов'язково різні. ***Степенем*** множини А називається його n-кратний прямий добуток самого на себе:

$$A^n = \underbrace{A \times \ldots \times A}_{n}.$$

Наслідок: $|A^n| = |A|^n$.

2.2.2. Бінарні відношення

Прості бінарні відношення вже розглядались у зв'язку з вивченням математичних структур. Далі відношення розглядаються детальніше. Нехай A і B — дві множини. ***Бінарним відношенням*** чи ***відповідністю*** між множинами A і B називається структура $\langle A, B; R\rangle$, де R — підмножина прямого добутку A і B:

$$R \subset A \times B.$$

R називається ***графіком*** відношення, A називається ***областю відправлення***, а B — ***областю прибуття***. Якщо множини A і B визначені контекстом, то часто просто кажуть, що задано відношення R.

Серед елементів множин A і B можуть бути такі, які не входять ні в одну з пар, що визначають відповідність R. Множина елементів області відправлення A, які присутні в парах відповідностей, називається ***областю визначення.*** Множина елементів області прибуття B, які присутні в парах відповідностей, називається ***областю значень*** (Рис. 2.9).

Якщо маємо багато пар $(a, b) \in R$, то для даного a усі відповідні елементи b називаються ***образами,*** а для даного b усі відповідні елементи a називаються ***прообразами.***

Якщо $\forall\ (a, b) \in R\ (a \in A)$, то відповідність R називається ***всюди*** або ***повністю визначеною***. Інакше відповідність називається ***частковою***.

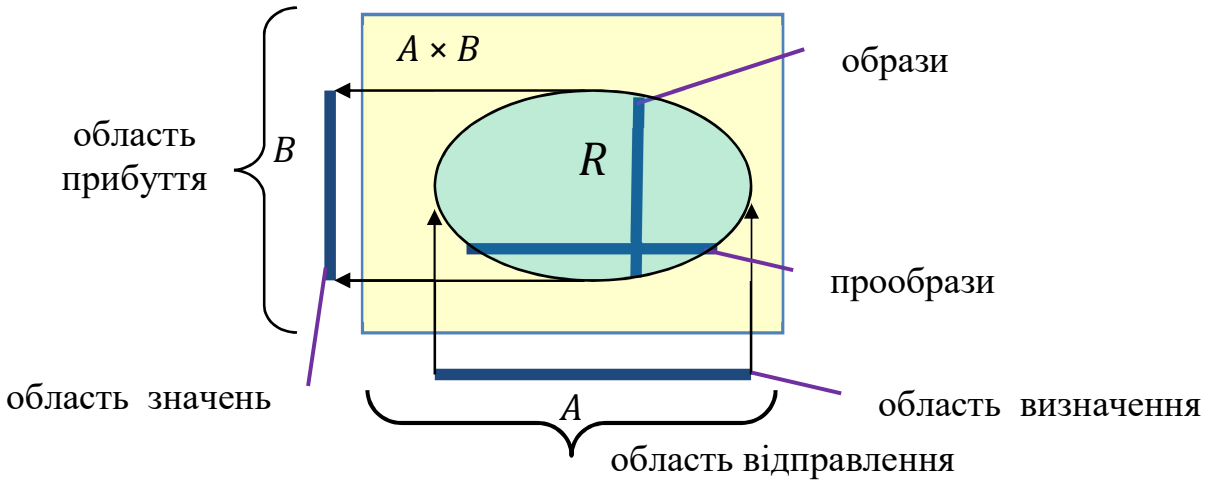
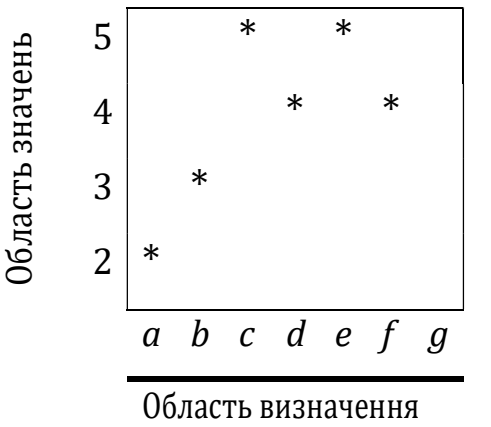
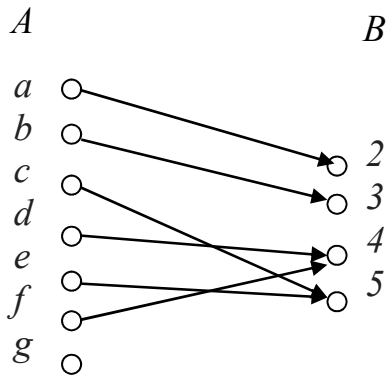


Рис. 2.9. Бінарне відношення між множинами *A* і *B*

Приклад. Нехай *A* = {*a, b, c, d, e, f, g*} — множина студентів, *B* ={2, 3, 4, 5} — оцінки на контрольній. Тоді графік відповідності у формі, власне, графіка, дводольного графа, таблиці та матриці показані на Рис. 2. а, б, в і г, відповідно.



а)



б)

	a	*b*	*c*	*d*	*e*	*f*	*g*
2	1						
3		1					
4				1		1	
5			1		1		

в)

$$\begin{pmatrix} 1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0 \\ 0&0&1&0&1&0&0 \\ 0&0&1&0&1&0&0 \end{pmatrix}$$

г)

Рис. 2.10. Способи представлення відношень

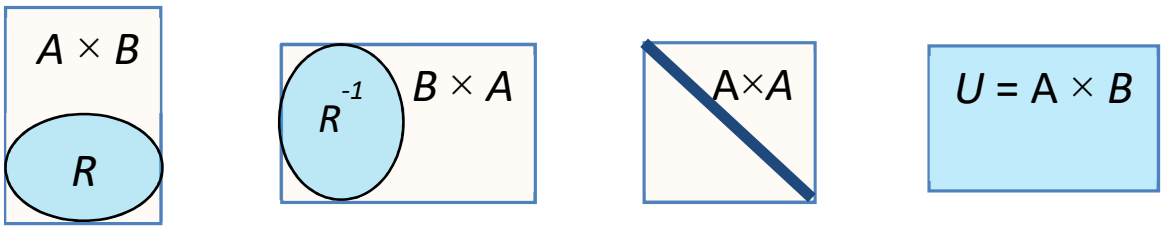
Існують наступні види відношень:

$R^{-1} \stackrel{\text{def}}{=} \{(b, a) \mid (a, b) \in R\} \subset B \times A$ — ***зворотне, інверсне відношення***;

$I \stackrel{\text{def}}{=} \{(a, a) \mid a \in A\} \subset A \times A$ — ***тотожність***;

$U \stackrel{\text{def}}{=} \{(a, b) \mid a \in A \vee b \in B\} = A \times B$ — ***універсальне відношення***.

Рис. 2.11 іллюструє ці відношення.



а) б) в) г)

Рис. 2.11. Відношення (а), його інверсне відношення (б), відношення тотожності (в) і універсальне відношення (г)

Аналогічно до впорядкованих множин, вводиться ***n-арне відношення***:

$R \subset A_1 \times ,..., \times A_n \stackrel{\text{def}}{=} \{(a_1,..., a_n) \mid a_1 \in A_1 \wedge ... \wedge a_n \in A_n\}$.

Вище було показано, що тернарне відношення — це основа для подання алгебраїчних операцій.

2.2.3. Операції з відношеннями

Оскільки бінарні відношення визначаються як множини елементів над добутком $A \times B$, то над ними можна виконувати операції об'єднання, перетину і доповнення. Якщо R_1, R_2 — два бінарних відношення, то їх ***об'єднання*** визначається як (Рис. 2.12):

$R = R_1 \cup R_2 \Leftrightarrow \forall a \in A, \forall b \in B\ (a\ R_1 b \vee a\ R_1 b)$.

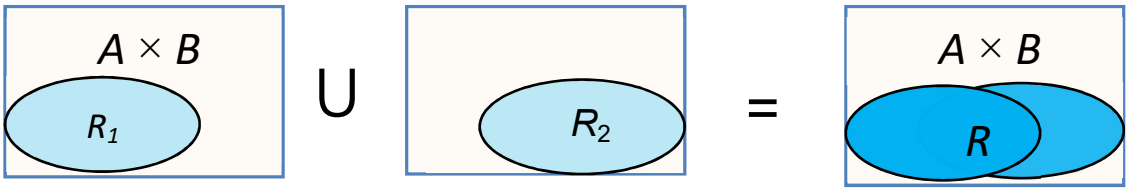


Рис. 2.12. Об’єднання відношень

Відношення ***доповнення*** визначається як:

$$\bar{R} \stackrel{\text{def}}{=} \{(a, b) \mid (a, b) \notin R\} \subset A \times B.$$

Композиція відношень

Нехай $R_1 \subset A \times C$ — відношення між A та C, а $R_2 \subset C \times B$ — відношення між C та B. ***Композицією*** двох відношень R_1 та R_2 називається відношення $R \subset A \times B$ між A та B таке, що:

$$R = R_1 \circ R_2 \stackrel{\text{def}}{=} \{(a, b) \mid a \in A \wedge b \in B \wedge \exists c \in C\ (aR_1c \wedge cR_2b)\}.$$

Іншим чином це записується як:

$$aR_1 \circ R_2b \Leftrightarrow \exists c \in C\ (aR_1c \wedge cR_2b).$$

Графічно композиція відношень зображається за допомогою відношення тотожності, яке перенапрямлює результат одного відношення на область відправлення іншого відношення (Рис. 2.13, а). Представлення композиції відношень об’єднанням дводольних графів є більш наочним (Рис. 2.13, б).

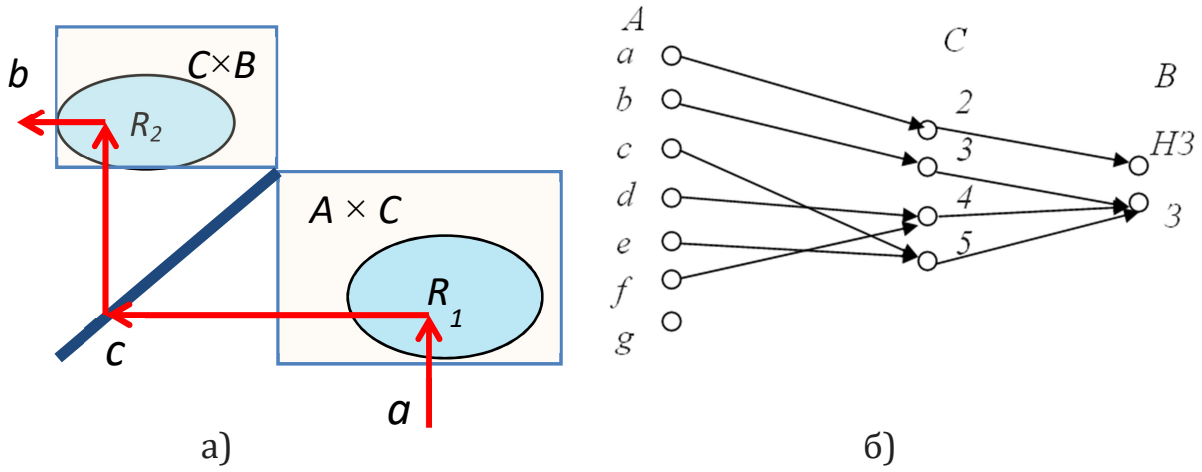
Композиція відношень — асоціативна, тобто

$$R_1 \circ (R_2 \circ R_3) = (R_1 \circ R_2) \circ R_3.$$

Але у загальному випадку, композиція відношень не комутативна, тобто,

$$R_1 \circ R_2 \neq R_2 \circ R_1.$$

Цей факт є очевидним після огляду Рис. 2.13.



а) б)

Рис. 2.13. Композиція відношень $R_1 \subset A \times C$ та $R_2 \subset C \times B$

Степінь відношень

Нехай R — відношення на множині A. ***Степенем*** відношення R називається його n-кратна композиція з самим собою і позначається як

$$R^n = \underbrace{R \circ \ldots \circ R}_{n}.$$

Приклад. Є відношення R циклічної перестановки чотирьох предметів. Знайти четверту степінь цього відношення. На Рис. 2.14 показані графи R та $R \circ R \circ R \circ R = R^4$. Можна прослідкувати, що у даному прикладі в результаті одержане відношення тотожності.

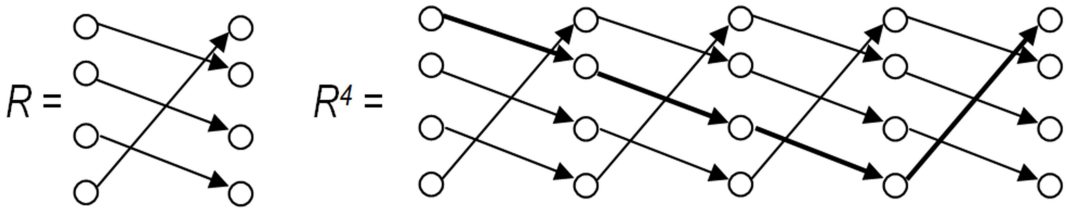


Рис. 2.14. Відношеня R та його четвертий степінь

2.2.4. Властивості відношень

Відношення R на множині A називається

— ***рефлексивним***, якщо $\forall a \in A$ $(a\ R\ a)$;

— ***антирефлексивним***, якщо $\forall a \in A$ $(\bar{a}\ R\ a)$;

— ***симетричним***, якщо $\forall a, b \in A\ (a\ R\ b \Rightarrow b\ R\ a)$;

— ***антисиметричним***, якщо $\forall a, b \in A\ (a\ R\ b \wedge b\ R\ a \Rightarrow a = b)$;

— ***транзитивним***, якщо $\forall a, b, c \in A\ (a\ R\ b \wedge b\ R\ c \Rightarrow a\ R\ c)$;

Рефлексивне, симетричне та транзитивне відношення називається ***відношенням еквівалентності*** (або просто ***еквівалентністю***).

Рефлексивне, антисиметричне та транзитивне відношення називається ***відношенням порядку*** (чи просто ***порядком***).

Відношення є ***лінійним чи повним,*** якщо

$\forall a, b \in A\ (a = b \Rightarrow a\ R\ b \vee b\ R\ a)$;

Для відношень $R \subset A \times A$ можна довести наступні теореми:

Т1: R рефлексивне $\Leftrightarrow I \subset R$.

Т5: R антирефлексивне $\Leftrightarrow R \cap I = \varnothing$.

Т2: R симетричне $\Leftrightarrow R = R^{-1}$.

Т4: R антисиметричне $\Leftrightarrow R \cap R^{-1} \subset I$.

Т3: R транзитивне $\Leftrightarrow R \circ R \subseteq R$.

Т6: R лінійне $\Leftrightarrow R \cup I \cup R^{-1} = U$.

Ядро відношення

Для відношення $R \subset A \times B$ композиція $\ker R = R \circ R^{-1}$ називається ***ядром*** цього відношення. Ядро — це таке відношення $\ker R \subset A^2$. З визначення композиції та зворотного відношення випливає властивість ядра:

$a \ker R\ c \Leftrightarrow \exists b \in B\ (a\ R\ b \wedge c\ R\ b)$.

Прикладом ядра відношення “бути знайомим” є відношення „бути знайомим або мати загального знайомого”.

2.2.5. Відношення перестановки

Нехай маємо $M = \{1,2,3,4,5\}$ і кортеж $A = (a_1,...,a_5)$. Тоді відношенням ***перестановки*** (підстановки) $P \subset M \times M$ є відношення $x\,P\,y$, яке переставляє елемент a_x на місце елемента a_y. Таке відношення має матрицю типу:

$$P = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix},$$

і граф перестановки, як на Рис. 2.15. Тобто у кожному рядку і стовпці матриці стоїть по одній одиниці. Тут перший елемент переставляється на третє місце, другий — на п’яте і т. д.

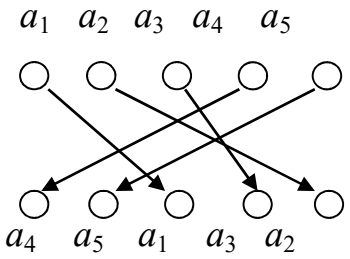


Рис. 2.15. Граф перестановки

Таке відношення можна закодувати матрицею підстановки:

$$P = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 4 & 1 & 2 \end{pmatrix},$$

Для відношення перестановки нескладно знайти інверсне відношення, переставивши рядки матриці підстановки і упорядкувавши її стовпці, як наприклад:

$$P^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 5 & 1 & 3 & 2 \end{pmatrix}.$$

Композиція прямого і інверсного відношень перестановки, як очікувалось, дорівнює відношенню тотожності:

$$P \circ P^{-1} = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 1\,2\,3\,4\,5 \end{pmatrix}.$$

Приклад відношення циклічної перестановки показано на Рис. 2.14.

Множина перестановок $P \in \mathcal{P}$ та операція їх композиції, яка називається ***множенням перестановок***, формують ***групу перестановок*** $\langle P; \circ \rangle$ (див. підрозд. 1.9).

2.2.6. Замикання відношень

Замикання є доволі загальним математичним поняттям. Неформально кажучи, властивість замкненості означає, що багаторазове виконання допустимих операцій не виводить математичну структуру за певні межі. Наприклад, це послідовність чисел із замкненого інтервалу, яка сходиться до межі, що належить цьому інтервалу.

Нехай R і R' — відношення на множині A. Відношення R' називається ***замиканням*** R щодо властивості C (транзитивне тощо), якщо:

— R' має властивість C, тобто $C(R')$ — істинне;

— R є підмножиною R': $R \subset R'$;

— відношення R' є найменшим з відношень з цією властивістю:

$R \subset R'' \wedge C(R'') \Rightarrow R' \subset R''$.

Транзитивне замикання визначається як об'єднання усіх можливих степенів даного відношення:

$R^{\circ} \stackrel{\text{def}}{=} R \cup R^2 \cup, \ldots, \cup R^{\infty}$.

Транзитивне і рефлексивне замикання визначається як:

$R^* \stackrel{\text{def}}{=} R^{\circ} \cup I$.

Якщо A — це множина людей (живих або мертвих), і — відношення «є батьком або матір'ю», тоді транзитивним замиканням R^* є відношення «є предком».

Згідно з визначеннями, матриця рефлексивного замикання одержується як об’єднання матриць усіх добутків відношення. У матриці транзитивного замикання додатково встановлюється одинична діагональ.

Приклад.

$$R = \begin{pmatrix} 1&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \\ 0&1&0&0 \end{pmatrix}, \quad R^2 = \begin{pmatrix} 1&1&1&0 \\ 0&0&0&1 \\ 0&1&0&0 \\ 0&0&1&0 \end{pmatrix}, \quad R^3 = \begin{pmatrix} 1&1&1&1 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{pmatrix},$$

$$R^\circ = R \cup R^2 \cup ,\ldots,\cup R^\infty = \begin{pmatrix} \mathbf{1}&1&1&1 \\ 0&\mathbf{1}&1&1 \\ 0&1&\mathbf{1}&1 \\ 0&1&1&\mathbf{1} \end{pmatrix} = R^*.$$

Одержали матрицю транзитивного замикання, у якої встановлена одинична діагональ. Отже, це є і рефлексивне замикання.

Отже, властивості замикання відношень нескладно виявити за їх матрицями. Рис. 2.16 ілюструє властивості замикань і елементи матриці, які виділені кольором, підкреслюють ці властивості.

транзитивне ($a \le b$)

$$R = \begin{pmatrix} 1&0&0&0 \\ 1&1&0&0 \\ 1&1&1&0 \\ 1&1&1&1 \end{pmatrix} \Leftrightarrow R \circ R = \begin{pmatrix} 1&0&0&0 \\ 1&1&0&0 \\ 1&1&1&0 \\ 1&1&1&1 \end{pmatrix} \subseteq R;$$

лінійне

$$R = \begin{pmatrix} 0&0&0&0 \\ \mathbf{1}&0&0&0 \\ \mathbf{1}&\mathbf{1}&0&0 \\ \mathbf{1}&\mathbf{1}&\mathbf{1}&0 \end{pmatrix} \Leftrightarrow R \cup I \cup R^{-1} = \begin{pmatrix} \mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1} \\ \mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1} \\ \mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1} \\ \mathbf{1}&\mathbf{1}&\mathbf{1}&\mathbf{1} \end{pmatrix} = U.$$

Рис. 2.16. Приклади замикань відношень

2.2.7. Подання відношень у програмах

Як було показано на Рис. 2.10, відношення зображається у вигляді булевої матриці. Нехай відношення $R \subset A^2$ зображене як булева матриця $[R]$. Аналогічно можна подати інші відношення. Тоді можна довести, що зворотне відношення одержується як транспонування матриці:

$$[R^{-1}] = [R]^T;$$

відношення доповнення є різницею матриць:

$$[\bar{R}] = [U] - [R];$$

а композиції відношень відповідає добуток матриць:

$$[R_1 \circ R_2] = B([R_1] \cdot [R_2]),$$

де $B(a_{i,j}) =$ у $= \begin{cases} 1 & \text{при } a_{i,j} > 0, \\ 0 & \text{при } a_{i,j} = 0. \end{cases}$

Причому, згідно з операцією добутку матриць, елемент результуючої матриці дорівнює сумі добутків і тому може бути більшим за одиницю. Тоді як результат береться булеве перетворення B добутку матриць, яке замінює будь-яке ненульове число на одиницю.

Приклад. Нехай A ={1,2,3,4}, $B = \{a, b, c\}$, $R = \begin{pmatrix} 0\,1\,1\,0 \\ 1\,1\,0\,0 \\ 1\,0\,1\,1 \end{pmatrix}$. Тоді зворотне відношення дорівнює $R^{-1} = \begin{pmatrix} 0\,1\,1 \\ 1\,1\,0 \\ 1\,0\,1 \\ 0\,0\,1 \end{pmatrix}$.

Відношення доповнення є різницею матриць:

$$[\overline{R}] = [U] - [R] = \begin{pmatrix} 1\,1\,1\,1 \\ 1\,1\,1\,1 \\ 1\,1\,1\,1 \end{pmatrix} - \begin{pmatrix} 0\,1\,1\,0 \\ 1\,1\,0\,0 \\ 1\,0\,1\,1 \end{pmatrix} = \begin{pmatrix} 1\,0\,0\,1 \\ 0\,0\,1\,1 \\ 0\,1\,0\,0 \end{pmatrix}.$$

Ядро відношення R дорівнює

$$\ker R = R \circ R^{-1} = \begin{pmatrix} 1\,1\,1 \\ 1\,1\,1 \\ 1\,1\,1 \end{pmatrix}.$$

Даний приклад відношення можна інтерпретувати як граф, вершини якого задані множиною B, а ребра – множиною A. У простій інтерпретації граф представляє план розміщення міст $\{a, b, c\}$ та доріг, які їх зв'язують (Рис. 2.17). Зважаючи на те, що ядро відношення представляє собою відношення типу "бути сусідом або мати загального сусіда", то заповнена одиницями матриця ker R свідчить про те, що у прикладі усі міста зв'язані одне з одним.

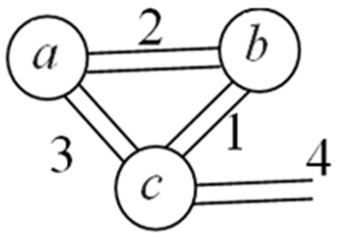


Рис 2.17. Приклад інтерпретації відношення R

Якщо множина A, на якій задається відношення є великою, то матрицю відношення стискають тим чи іншим способом. Наприклад, колонки матриці кодують як множину у вигляді бітової шкали. Також застосовують хеш-таблиці.

Властивості відношень нескладно виявити за їх матрицями. Рис. 2.18 ілюструє певні властивості відношень і елементи матриці, які виділені кольором, підкреслюють ці властивості.

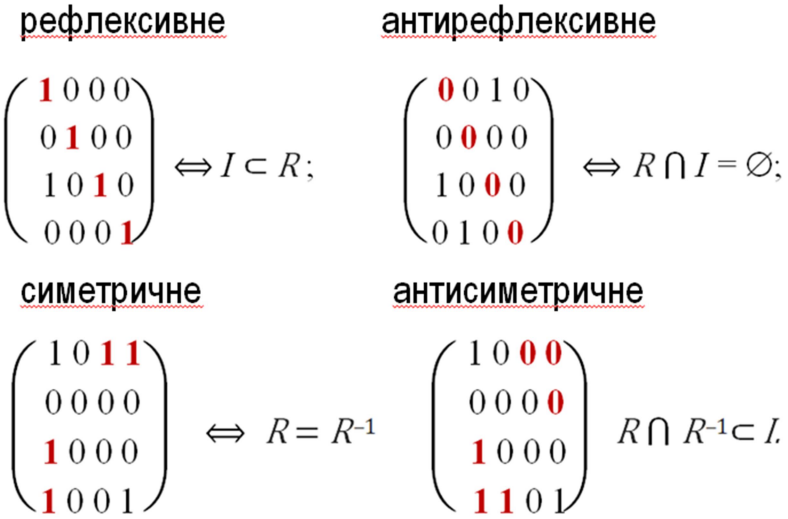


Рис 2.18. Приклади властивостей відношень

n-арні відношення використовуються, наприклад, в теорії баз даних. Сама назва «реляційна» база даних походить від слова relation (відношення). Реляційна база складається з кортежів або записів. Приклад такого запису показано на Рис 2.19.

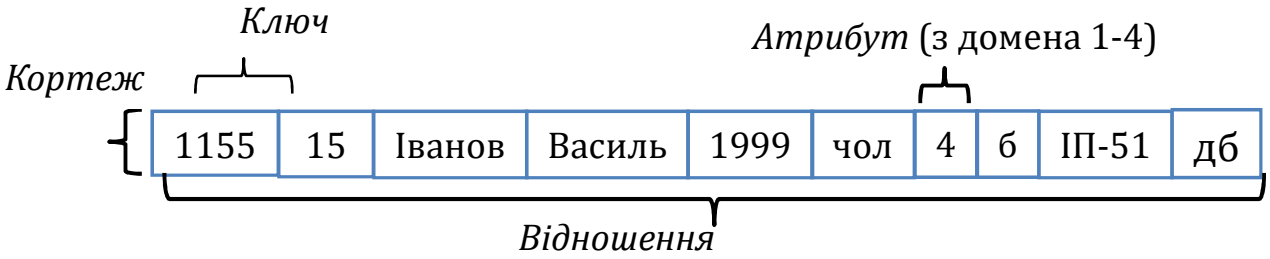


Рис. 2.19. Приклад запису бази даних як 10-арного відношення

2.2.8. Завдання

1. Визначте, які з властивостей (рефлексивність, симетричність, антисиметричність, транзитивність) мають наступні відношення на множині {1, 2, 3, 4, 5}:

R_1: $aR_1b \leftrightarrow |a - b| = 1$;

R_2: $aR_2b \leftrightarrow 0 < a - b < 3$;

R_3: $aR_3b \leftrightarrow a + b$ — парне число;

R_4: $aR_4b \leftrightarrow a \geq b^2$;

R_5: $aR_5b \leftrightarrow$ НСД(a, b) =1.

2. Визначте які наступні твердження коректні:

1) будь-яке відношення на множині або симетричне, або антисиметричне;

2) жодне відношення не може бути одночасно симетричним та антисиметричним;

3) якщо відношення R є рефлексивним, симетричним, антисиметричним або транзитивним то і R^{-1} також має таку саму властивість.

3. Між множинами $A = \{a, b, c, d, e\}$ і $B = \{1, 2, 3, 4, 5\}$ задані відповідності *Ri.* Визначити, які з цих відповідностей усюди визначені, функціональні, ін'єктивні, сюр'єктивні, бієктивні.

1) $R_1 = \{(a, 2), (a, 5), (b, 1), (b, 5), (c, 2), (d, 3), (d, 5)\}$,

2) $R_2 = \{(a, 3), (b, 2), (c, 3), (e, 3)\}$,

3) $R_3 = \{(a, 2), (b, 3), (c, 4), (d, 1), (e, 5)\}$,

4) $R_4 = \{(a, 2), (a, 3), (b, 1), (c, 4), (c, 5), (d, 1), (e, 2), (e, 4)\}$,

5) $R_5 = \{(a, 4), (b, 3), (c, 5), (d, 2), (e, 1)\}$.

4. Встановити властивості відношення $R \subset A \times B$, $A = \{1,2,...8\}$, $B = \{1, 2,...,15\}$, $(a,b) \in R$, у тому числі, чи є відношення функціональним.

1) $a < b$;

2) $a > b$;

3) $a^2 + b^2 > ab$;

4) $a - b < 3$;

5) a є дільником b;

6) $b - a > 6$;

7) $a^2 + b^2 > ab$;

8) $a^2 = b$;

9) $|a - 4| > |b - 6|$;

10) $2a = b$;

11) $|b - a| > 5$;

12) $|a - b| > 5$;

13) $a + b$ ділиться на 3.

5. Які з наступних відношень на множині цілих чисел $\mathbb{Z}$ є відношеннями порядку:

R_1: $aR_1b \leftrightarrow a \leq b$;

R_2: $aR_2b \leftrightarrow a \geq b$;

R_3: $aR_3b \leftrightarrow a < b$;

R_4: $aR_4b \leftrightarrow a^2 \geq b^2$;

R_5: $aR_5b \leftrightarrow a = b$;

R_6: $aR_6b \leftrightarrow a$ ділиться на b ;

R_7: $aR_7b \leftrightarrow НЗД\ (a, b) = 1$?

2.2.6. Маємо відношення перестановки $P = \begin{pmatrix} 1\ 2\ 3\ 4\ 5 \\ 5\ 1\ 4\ 3\ 2 \end{pmatrix}$.

1) Знайти P^{-1}.

2) Знайти P^2, P^3, P^4, P^5, P^6.

3) Знайти транзитивне замикання P.

4) Знайти транзитивне і рефлексивне замикання P.

2.3. Функції

2.3.1. Поняття функції

Поняття функції — це одне з основних понять у математиці. У даному підрозділі маються на увазі функції, які відображають об'єкти однієї множини у об'єкти іншої.

Бінарне відношення f називається ***функцією***, якщо

$\forall a\,((a, b) \in f \wedge (a, c) \in f \Rightarrow b = c)$,

тобто, якщо є відношення з образом a, то воно єдине (Рис. 2.20). Така властивість цього відношення називається ***однозначністю*** або ***функціональністю***.

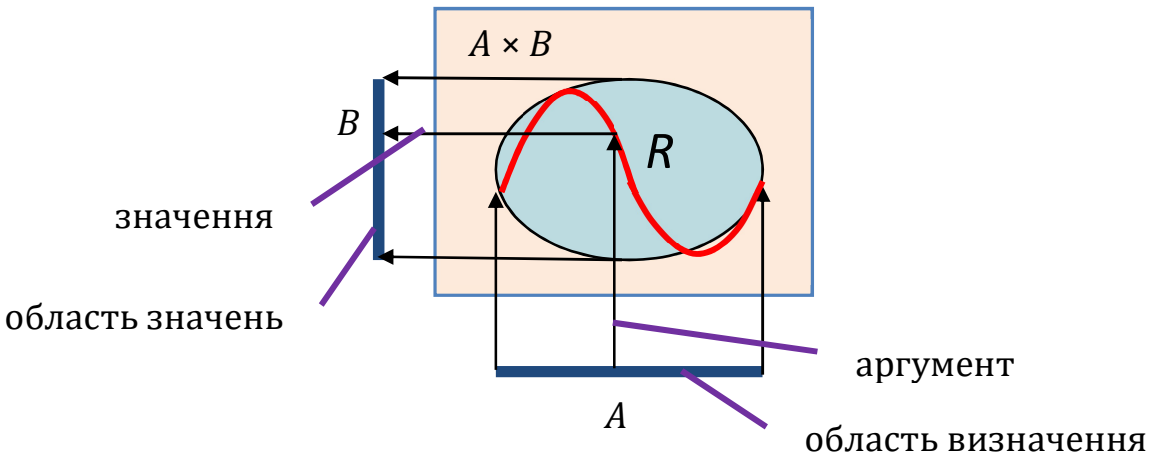


Рис. 2.20. Функція як відношення

Функцію записують одним з наступних способів:

$$f\colon A \to B;\ A \xrightarrow{f} B;\ b = f(a);\ (a,b) \in f.$$

Тут a називають ***аргументом***, а b — ***значенням*** функції. Виходячи з того, що функція — це різновид відношення, то множина A називається ***областю визначення***, а B — ***областю значень***. Якщо функція f скрізь

визначена і області значень та визначення співпадають, то f: $A \to A$ називають ***перетворенням*** над A.

В математиці, як правило, розглядаються ***тотальні*** функції, тобто такі, що визначені скрізь. Програмісти ж мають справу з ***частковими*** функціями, які визначені не для всіх допустимих значень. Наприклад, математична функція $\sqrt{x}$ визначена для всіх x, а стандартна функція мови програмування sqrt дає правильний результат не для всіх можливих значень аргументу, принаймні, тільки для позитивних і нуля.

Функція f: $A_1 \times ,..., \times A_n \to B$ називається функцією від n аргументів, або n-місною функцією. Така функція відображає кортеж у елемент множини області визначення, тобто $b = f(a_1,..., a_n)$. На Рис. 2.21 показана функція від двох аргументів $b = f(a_1, a_2)$.

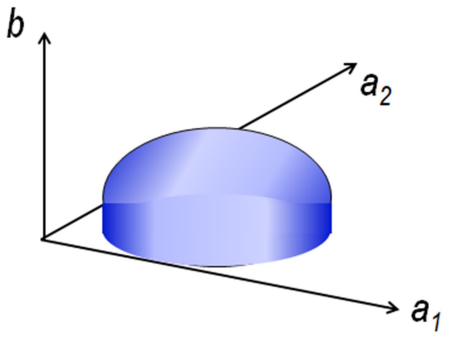


Рис. 2.21. Функція від двох аргументів

2.3.2. Ін'єкція, сюр'єкція і бієкція

Функція f: $A \to B$ називається:

ін'єктивною або ***ін'єкцією***, якщо $b = f(a_1) \wedge b = f(a_2) \Rightarrow a_1 = a_2$;

сюр'єктивною або ***сюр'єкцією***, якщо $\forall b \in B(\exists\, a \in A\ (b = f(a_2))$;

і бієктивною або ***бієкція***, якщо $\forall a \in A((a,b) \in f \wedge (a,c) \in f \Rightarrow b = c)$; тобто, якщо вона ін'єктивна та сюр'єктивна.

Рис. 2.22 ілюструє ці види функціональних залежностей.

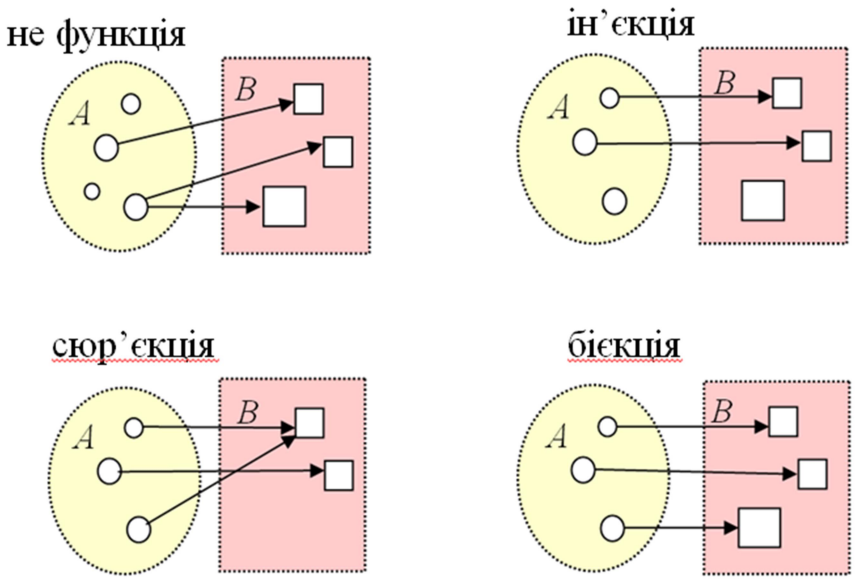


Рис. 2.22. Функціональні залежності: відношення, але не функція, ін'єкція, сюр'єкція та бієкція

На Рис. 2.23 показані приклади відношення, але не функції, функцій ін'єкції, сюр'єкції та бієкції. Слід зазначити, що якщо ми маємо на увазі функції дискретної математики, то в данному прикладі повинні бути аргументи і функції, які представлені раціональними числами.

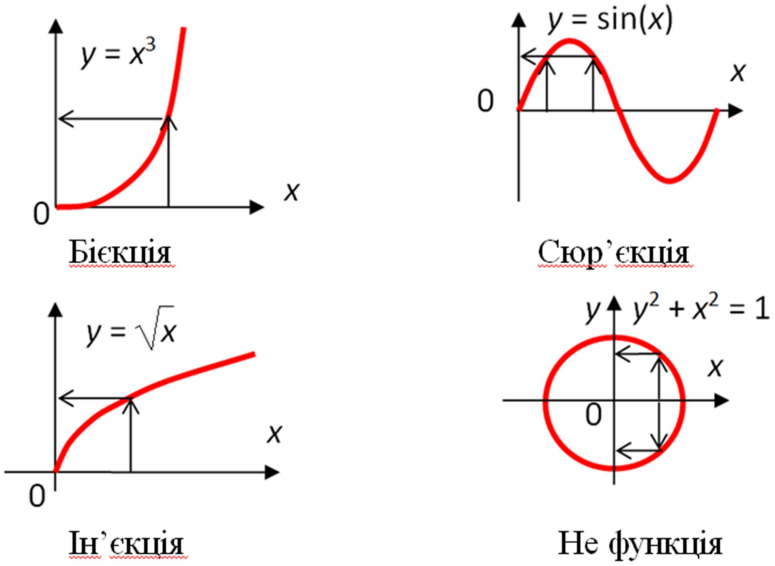


Рис. 2.23. Приклади функціональних залежностей

Теорема. Бієктивна функція $f: A \to B$ має зворотну функцію $f^{-1}: B \to A$. Рис. 2.22 ілюструє цю теорему.

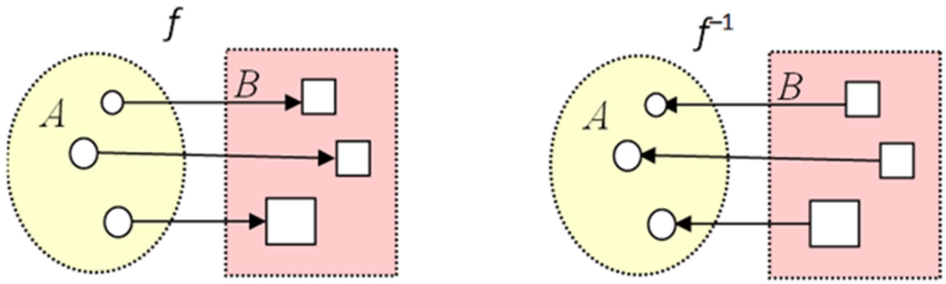


Рис. 2.24. Бієктивна функція f та її зворотна функція f^{-1}.

2.3.3. Суперпозиція функцій

Суперпозиція функцій — це композиція відношень, що відповідають цим функціям. Суперпозиція функцій визначається як:

$(f \circ g)(x) \stackrel{\text{def}}{=} f(g(x))$.

Неважко довести теорему:

$f: A \to B \wedge g: B \to C. \Rightarrow (g \circ f): A \to C,$

тобто суперпозиція функцій також є функцією. Рис. 2.25 ілюструє суперпозицію функцій $\sqrt{x}$ та z^3.

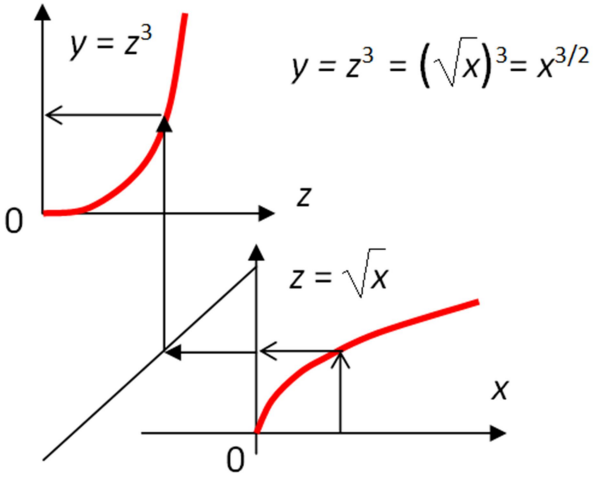


Рис. 2.25. Приклад суперпозиції функцій

2.3.4. Подання функцій у програмах

Нехай f: $A \to B$, а множина A є скінченною з потужністю $|A| = n$, яка не переважає кількох тисяч. Найбільш загальним зображенням такої функції є масив `array [A] of B`, де `A` — тип даних, значення якого зображають елементи множини A, а `B` — тип даних, значення якого відповідають множині B. У більшості випадків доступ до елемента масиву виконують за допомогою номера цього елементу, тобто елементи множини A нумеруються і функція зображається за допомогою масиву `array [1..`n`] of B`.

Функція кількох аргументів зображається багатовимірним масивом.

Якщо множина A велика, то використання масивів для подання функцій є неефективним з точки зору економії пам'яті. У такому випадку для введення функцій використовується особливий вид процедур, які повертають єдине значення для заданого значення аргументу. Зазвичай такі процедури також називаються «функціями». У деяких мовах програмування визначення функцій вводяться ключовим словом `function`.

Багатомісні функції зображуються за допомогою декількох формальних параметрів у визначенні функції. Властивість функціональності забезпечується оператором повернення з підпрограми. Цей оператор часто позначається ключовим словом `return`, яке припиняє виконання тіла функції і одночасно повертає значення.

При поданні функції виконується мінімізація її складності обчислення тим чи іншим методом її апроксимації та табулюванння. В решті-решт, теорія алгоритмів зародилась як результат вирішення проблеми обчисленності функцій.

2.3.5. Завдання

1 Між множинами A = {a, b, c, d, e} і B = {1, 2, 3, 4, 5} задані відповідності Ri. Визначити, які з цих відповідностей усюди визначені, функціональні, ін'єктивні, сюр'єктивні, бієктивні.

1) R1 = {(a, 2), (a, 5), (b, 1), (b, 5), (c, 2), (d, 3), (d, 5)},

2) R2 = {(a, 3), (b, 2), (c, 3), (e, 3)},

3) R3 = {(a, 2), (b, 3), (c, 4), (d, 1), (e, 5)},

4) R4 = {(a, 2), (a, 3), (b, 1), (c, 4), (c, 5), (d, 1), (e, 2), (e, 4)},

5) R5 = {(a, 4), (b, 3), (c, 5), (d, 2), (e, 1)}.

2. Встановити властивості відношення $R \subset A \times B$, A = {1,2,…8}, B = {1, 2,…,15}, $(a,b) \in R$, у тому числі, чи є відношення функціональним.

1) $a < b$;

2) $a > b$;

3) $a2 + b2 > ab$;

4) $a - b < 3$;

5) a є дільником b;

6) $b - a > 6$;

7) $a2 + b2 > ab$;

8) $a2 = b$;

9) $|a - 4| > |b - 6|$;

10) $2a = b$;

11) $|b - a| > 5$;

12) $|a - b| > 5$;

13) a + b ділиться на 3.

2.4. Відношення порядку і ґратки

2.4.1. Відношення порядку

Над елементами множини M можна задати бінарне ***відношення порядку*** або ***порядок*** «$\prec$». Для множини цілих чисел воно еквівалентне відношенню «$\leq$».

Відношення порядку задовольняє наступним властивостям:

P_1: $\forall\, a \in M (a \prec a)$ (рефлективність);

P_2: $\forall\, a, b \in M (a \prec b \wedge b \prec a \Rightarrow a = b)$ (антисиметричність);

P_3: $\forall\, a, b, c \in M (a \prec b \wedge b \prec c \Rightarrow a \prec c)$ (транзитивність).

Порядок є ***досконалим*** або ***лінійним***, якщо

$$C_1: \forall\, a, b \in M (a \prec b \vee b \prec a),$$

тобто, між будь-якими елементами множини є відношення порядку, інакше — це ***частковий порядок***.

Отже, у лінійному порядку, якщо розглядаються будь-які два елементи множини, то один з них більший за іншого.

Приклад множини з лінійним порядком — це колектив, де порівнюють його членів за віком. Частковий порядок виникає тоді, коли не для всіх пар елементів множини задано відношення порядку. Наприклад, це робочий колектив, де порядок встановлюється за правилом: начальник — підлеглий, тобто знаходяться пари людей, серед яких немає такого відношення.

Приклад: Нехай маємо множину

$$A = \{a, b, c, d, e, f\} \sim \{3, 4, 12, 24, 48, 72\},$$

яка являє собою деякі цілі числа. Нехай відношення $\prec$ означає, що при $x \prec y$ число x є дільником націло числа y. Тоді маємо множину відношень

$$\{a \prec a, a \prec c, b \prec b, b \prec c, c \prec c, c \prec d, d \prec d, d \prec e, d \prec f, e \prec e, f \prec f\}.$$

Множину, на якій задано відношення порядку, традиційно зображають графом, який називають ***діаграмою Хасе.*** Такий граф для множини з прикладу вище показано на Рис. 2.26. Традиційно, у діаграмах Хасе вершини, що стоять у рисунку нижче, передують за відношенням «$\prec$» ті вершини, що стоять вище.

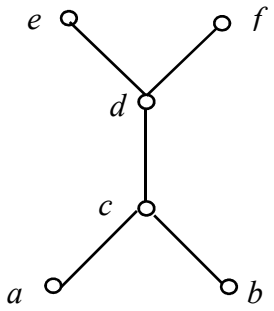


Рис. 2.26. Приклад діаграми Хасе

На упорядкованій множині можна визначити наступні функції, які характеризують цю множину.

$\Gamma = \mathrm{Max}(a, b)$ — це ***максимум*** a і b, якщо $a \prec \Gamma \wedge b \prec \Gamma$.

$\Delta = \mathrm{Min}(a, b)$ — це ***мінімум*** a і b, якщо $\Delta \prec a \wedge \Delta \prec b$.

Якщо порядок частковий, то таких максимумів і мінімумів може бути багато. Наприклад, на Рис.2.22 $\mathrm{Max}(c, d) = e$ і $\mathrm{Max}(c, d) = f$.

Строгий максимум γ для елементів a, b це елемент $\gamma = \sup(a, b)$, якщо $\forall\Gamma\ (\gamma \prec \Gamma)$.

Строгий мінімум δ для елементів a, b це елемент $\delta = \inf(a, b)$, якщо $\forall\Delta\ (\Delta \prec \delta)$.

Приклад. Для пари (c, d) на Рис.2.26 знайти максимуми та мінімуми: $\Gamma \in \{d, e, f\}$, $\Delta \in \{a, b, c\}$, $\sup(c, d) = d$, $\inf(c, d) = c$.

Але для пари (a, b) строгого мінімуму не існує, а $\sup(a, b) = c$. Тобто, тут порядок не є лінійний, бо $a \not\prec b$ і не є строгий, бо максимумів та мінімумів кілька. Тільки для лінійного порядку, для будь-яких a і b, якщо $a \prec b$, то $\sup(a, b) = b$, а $\inf(a, b) = a$.

Функції максимуму і мінімуму можуть бути також одно- чи багатомісні.

2.4.2. Ґратка

Ґратка — це структура **L** = ⟨*L*; ≺⟩, для якої виконуються аксіоми P_1, P_2, P_3, тобто аксіоми рефлексивності, антисиметричності та транзитивності а також аксіома:

P_4: $\forall\, a, b \in L\ \exists\ \sup(a, b) \wedge \inf(a, b)$,

тобто між елементами структури типу ґратка існує лише строгий порядок.

Приклад. Маємо множину цілих чисел, які є дільниками числа 36: D_{36} = {1, 2, 3, 4, 6, 9, 12, 18, 36} і відношення порядку: *a є дільником b*. Ці множина і відношення формують ґратку ⟨D_{36}; ≺⟩ з графом Хасе як на Рис. 2.27, а. Тут для будь-якої пари елементів є sup(*a, b*) і inf(*a, b*). Наприклад, ґратка ⟨*L*; ≺⟩ є ***лінійною***, якщо порядок "≺" — лінійний, тобто $\forall a, b \in L(a \prec b \vee b \prec a)$.

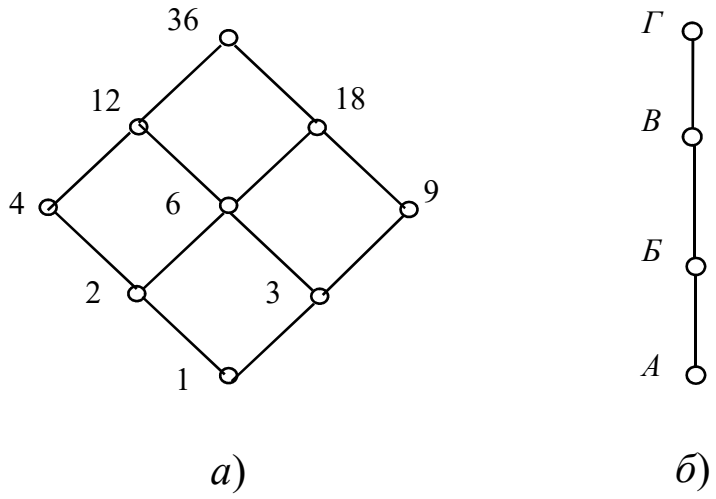


Рис. 2.27 Діаграми Хасе ґраток

Нехай маємо структуру ґратки ⟨*L*; ≺ ⟩, *L* ={ *А, Б, В, Г* }, причому *А* ≺ *Б*, *Б* ≺ *В* та *В* ≺ *Г*. Наприклад, *А* ≺ *Б* означає, що учень *Б* вищий за зростом за учня *А*. Тоді така ґратка є лінійною і зображується графом на Рис. 2.27, б.

Тут можна порівняти будь-яку пару елементів, наприклад, $A \prec B$. У ґратці на Рис. 2.27, а не всяку пару елементів можна порівняти, наприклад, $4 \not\prec 6$, тому така ґратка не є лінійною.

В дискретній математиці розвинута алгебра ґраток. Для зв'язку з алгебрами, операцію $\sup(a, b)$ іноді називають ***ґратчастим додаванням***, а операцію $\inf(a, b)$ — ***ґратчастим множенням***. Тоді для $a \prec b$ наступні пари: відношення — рівняння є еквівалентними.

$$\sup(a, b) \Leftrightarrow a \vee b = b;$$

$$\inf(a, b) \Leftrightarrow a \wedge b = a,$$

де символи «$\vee$» та «$\wedge$» означають операції ґратчастого додавання і множення. Ці операції, на відміну від арифметичних додавання і множення, мають наступні властивості:

1. Ідемпотентність:

$a \wedge a = a$, $a \vee a = a$.

2. Комутативність:

$a \wedge b = b \wedge a$.

3. Асоціативність:

$a \wedge (b \wedge c) = (a \wedge b) \wedge c$; $a \vee (b \vee c) = (a \vee b) \vee c$.

4. Поглинання:

$(a \wedge b) \vee a = a$; $(a \vee b) \wedge a = a$;

5. Ґратка називається дистрибутивною, якщо

$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$; $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$;

Якщо в ґратці є елемент $0 \in M$, такий що

$\forall a(0 \wedge a = 0$ та $1 \vee a = a)$,

то такий елемент називається ***нулем*** або ***нижньою гранню***. Аналогічно, якщо є елемент $1 \in M$, такий що

$$\forall a\ (1 \wedge a = a \text{ та } 1 \vee a = 1),$$

то такий елемент називається ***одиницею*** або ***верхньою гранню***.

Ґратка з верхньою та нижньою гранями називається ***обмеженою***. Приклад такої ґратки показаний на Рис. 2.27, а. У неї елемент 36 є одиницею, а 1 — нулем. Для елемента a знайдеться ***доповнюючий елемент*** $\bar{a}$, такий що

$$a \vee \bar{a} = 1,\ a \wedge \bar{a} = 0.$$

Причому, доповнюючих елементів може бути один, кілька або жодного.

Приклад. Знайти доповнюючі елементи для елементів 2 і 4, а також нульовий і одиничний елементи на Рис. 2.27, а.

Рішення. Для елемента 2 доповнюючими елементами є елементи 12 і 18, а для елементу 4 доповнюючого елементу немає, тому що $\forall x \neq 4 (4 \wedge x \neq 0 \text{ та } 4 \vee x \neq 1)$. Нульовим елементом є 1, а одиничним — 36.

На Рис. 2.27, б елементи А і Г є нулем і одиницею, А, Б взаємно доповнюють елементи В, Г, відповідно. Така обмежена ґратка, у якої у кожного елемента є доповнюючий, називається ***доповненою ґраткою***.

Важливим розвитком теорії ґраток є булева алгебра.

Приклад. Для множини {a,b,c,d,e,f} граткою буде наступне відношення

$\{a \prec a, a \prec b, b \prec b, b \prec d, c \prec c, c \prec e, d \prec d, d \prec f, e \prec e, e \prec f, f \prec f\}$.

Тут у кожної пари елементів відношення знайдеться максимум та мінімум.

2.4.3. Лексикографічний порядок

Нехай $\langle A; \leq\rangle$, $\langle B; \leq\rangle$ — частково впорядковані множини. Можна задати інший частковий порядок "$\prec$" на добуткові $A \times B$, який виражається наступним чином:

$$(a, b) \prec (a', b') \text{ якщо } a < a' \text{ або якщо } a = a' \wedge b \leq b'.$$

Порядок "$\prec$" називається ***лексикографічним*** (словниковим). Тут порівнюються два вектори. Причому старша координата є домінуючою. Тобто той вектор більший, у якого старша координата більша. Якщо ці координати рівні одна одній, то порівнюють молодші координати. Так само лексикографічно порівнюють довші вектори.

Нехай $\langle A_1; \leq\rangle, \ldots, \langle A_n ; \leq\rangle$ — частково впорядковані множини. Тоді лексикографічний порядок на множині $A_1 \times \ldots \times A_n$ задається наступним чином:

$$(a_1, a_2, \ldots, a_n) \prec (a_1', a_2', \ldots, a_n') \text{ якщо}$$
$$a_1 < a_1' \text{ або}$$
$$a_1 = a_1' \wedge a_2 < a_2' \text{ або}$$
$$a_1 = a_1' \wedge a_2 = a_2' \wedge \ldots \wedge a_n \leq a_n'.$$

Порядок появи слова в українському словнику є прикладом лексикографічного порядку.

Приклад: Нехай A = (а, б, в, ... я), і над A заданий звичайний лінійний порядок (а ≤ б), (б ≤ в), ... (ю ≤ я). Множина $A^n = A_1 \times \ldots \times A_n$ являє собою множину усіх слів, що мають довжину n. Тоді можна побудувати множину S^n упорядкованих пар (w_1, w_2) $(w_1, w_2 \in A^n | \ w_1 < w_2)$. Тобто w_1 передує w_2 у віртуальному словнику.

Таким чином, прикладами лексикографічної нерівності є

Бак $\prec$ бар, кит $\prec$ кіт, бук $\prec$ дуб.

Лексикографічний порядок використовується у комп'ютерах для сортування символьних даних. Такий порядок також називається

словниковим порядком. Використовуються назви "лексично менший" або "лексично рівні" чи "лексично більший" для позначення лексикографічної нерівності.

Елементи багатомірних масивів у комп'ютері індексують набором індексів $[a_1, a_2, ..., a_n]$, причому насправді, ці елементи розташовані у одновимірному масиві розміром $|A_1 \times ... \times A_n|$. Щоб визначити, котрий з пари елементів масиву з індексами $[a_1, a_2, ..., a_n]$ і $[a_1', a_2', ..., a_n']$ знаходиться за молодшими адресами у пам'яті, їх порівнюють лексикографічно.

2.4.4. Завдання

1. Для наступного відношення порядку побудувати діаграму Хасе і визначити, чи є множина з відношенням граткою:

1) $A = \{1, 2, 3, 4, 5, 6, 7, 8\}$, $R = \{ (x, y) \in A \times A, x \le y\}$.

2) $A = \{1, 2, 3, 5, 6, 10, 15, 30\}$, $R = \{ (x, y) \in A \times A, y$ ділиться на $x\}$.

3) $A = \{ 2, 3, 4, 6, 8, 9,12, 18, 24,36\}$, $R = \{ (x, y) \in A \times A, y$ ділиться на $x\}$.

2. Дослідити ґратку {a ≺ a, a ≺ b, b ≺ b, b ≺ d, c ≺ c, c ≺ e, d ≺ d, d ≺ f, e ≺ e, e ≺ f, f ≺ f } для множини {a,b,c,d,e,f}. Накреслити діаграму Хасе, визначити якого виду ґратка, визначити ґратчасте додавання і множення для усіх пар елементів множини, знайти доповнюючі елементи ґратки.

3. Словник має множину слів { бук, дуб, кит, кіт, бак, бар, баран}.

Слід відсортувати за лексикографічним порядком.

3. Булева алгебра

3.1. Визначення булевої алгебри

Дистрибутивна обмежена решітка, в якій для кожного елемента існує доповнюючий елемент, називається ***булевою алгеброю***. Виходячи з визначення доповнених ґраток, булева алгебра має наступні особливості:

1) $a \vee a = a$, $a \wedge a = a$.

2) $a \vee b = b \vee a$, $a \wedge b = b \wedge a$.

3) $a \wedge (b \wedge c) = (a \wedge b) \wedge c$; $a \vee (b \vee c) = (a \vee b) \vee c$.

4) $(a \wedge b) \vee a = a$, $(a \vee b) \wedge a = a$.

5) $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$; $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$;

6) $0 \wedge a = 0$, $0 \vee a = a$.

7) $1 \wedge a = a$, $1 \vee a = 1$.

8) $\bar{\bar{a}} = a$ (інволютивність).

9) $\overline{a \wedge b} = \bar{a} \vee \bar{b}$, $\overline{a \vee b} = \bar{a} \wedge \bar{b}$ (закони де Моргана).

10) $a \wedge \bar{a} = 0$, $a \vee \bar{a} = 1$.

Приклад булевої алгебри — структура $\langle 2^M; \wedge, \vee, \bar{} \rangle$, де 2^M — булеан, причому M — верхня грань, $\varnothing$ — нижня грань, а "$\prec$" — натуральний частковий порядок, тобто, $x \prec y$, якщо множина x на певний елемент менша за множину y. Для $M = \{a, b, c\}$ решітка булевої алгебри показана на Рис. 3.1. Тут кожен елемент має доповнюючий, наприклад, $ab = \bar{c}$, $b = \overline{ac}$, $abc = \bar{\varnothing} = 1$.

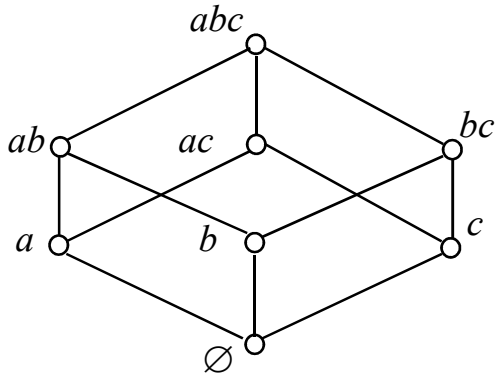


Рис. 3.1 Ґратка булевої алгебри

Традиційна булева алгебра – ***алгебра перемикальних функцій*** — це структура $\langle\{0,1\};\ \&,\ \vee,\ \bar{}\rangle$, де & — відношення кон'юнкції, $\vee$ — відношення диз'юнкції. Причому 0 — це нижня грань, 1 — верхня грань, а відношення $x \prec y$ — імплікація, тобто, коли $y = 1$ чи коли $x = y = 0$. Тут позначення 0 і 1 співпадають з нулем та одиницею ґратки, а сама ґратка є примітивною.

3.2. Функції алгебри логіки

3.2.1. Подання булевих функцій

Булева функція або ***функція алгебри логіки*** — це функція $f:\{0,1\}^n \to \{0,1\}$. Усі булеві функції від n аргументів формують множину булевих функцій P_n.

Функція f, яка залежить від n змінних x_1, x_2,..., x_n, називається ***булевою*** або ***перемикаючою***, якщо функція і будь-який з її аргументів x_i приймає значення тільки з множини $\{0, 1\}$. Аргументи x_i також називаються ***булевими***.

Будь-яка булева функцію задають одним з трьох способів: табличним, геометричним та аналітичним.

При табличному способі булева функція $f(x_1, x_2,..., x_n)$ задається ***таблицею істинності*** (табл. 3.1). У її лівій частині представлені усі можливі двійкові кортежі (набори) довжини n, а в правій — вказуються значення функції на цих наборах.

Табл. 3.1. Булева функція f від аргументів x_1,x_2,x_3,x_4 чи номеру набору N

x_1,x_2,x_3,x_4	N	F
0 0 0 0	0	1
0 0 0 1	1	0
0 0 1 0	2	0
0 0 1 1	3	1
0 1 0 0	4	1
0 1 0 1	5	0
0 1 1 0	6	1
0 1 1 1	7	1
0 0 0 0	8	1
0 0 0 1	9	0
0 0 0 0	10	0
0 0 0 1	11	0
0 0 0 0	12	0
0 0 0 1	13	0
0 0 0 0	14	1
0 0 0 1	15	0

Двійковий набір $\gamma = (\gamma_1, \gamma_2,...,\gamma_n)$, $\gamma_i \in \{0,1\}$ — це сукупність значень аргументів $x_1, x_2,..., x_n$ булевої функції f. Двійковий набір має довжину n, якщо він представлений n цифрами з множини {0,1}. У табл. 3.1 перелічені усі двійкові набори довжини 4.

Двійкові набори у таблиці істинності булевої функції зручно представляти номерами наборів. Якщо аргументи $x_1, x_2,..., x_n$ записані у порядку зростання індексів, тоді двійковий набір $\gamma = (\gamma_1, \gamma_2,...,\gamma_n)$ вважається цілим числом:

$$N = \gamma_1 \cdot 2^{n-1} + \gamma_2 \cdot 2^{n-2} + ... + \gamma_{n-1} \cdot 2 + \gamma_n,$$

яке називають ***номером набору*** γ**.** Наприклад, двійкові набори 0110 і 1011 мають номери 6 та 11, відповідно. Тоді функцію з табл. 3.1 можна записати як $f(0,3,4,6,7,8,14)$.

При геометричному способі булева функція $f(x_1, x_2,..., x_n)$ подається n-вимірним кубом. У геометричному сенсі, кожен двійковий набір $\gamma = (\gamma_1, \gamma_2,...,\gamma_n)$ представляє собою певний вектор, який визначає конкретну точку n-вимірного простору. Усі можливі двійкові набори формують вершини ***n-вимірного куба*** (***гіперкуба***). Подання булевої функції

геометричним способом полягає у маркуванні нулем чи одиницею вершин гіперкуба. На Рис. 3.2 показаний гіперкуб з закодованою функцією $f(1,2,4,7)$. Це функція суми за модулем 2 від трьох аргументів. Тут нульовий результат позначений крапкою, а одиничний — кружечком.

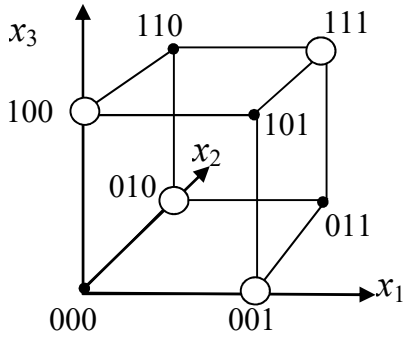


Рис. 3.2. Подання булевої функції геометричним способом

Булева функція задається аналітичним способом за допомогою формул, тобто, аналітичними виразами з використанням операцій булевої алгебри. Фактично, усі перетворення над булевими функціями, які необхідні для проектування цифрових автоматів, проводяться на аналітичному рівні.

Як показано вище, між двійковими наборами та двійковими числами є взаємно-однозначна відповідність. Отже, існують 2^n різних наборів двійкових змінних. Відповідно, областю визначення булевої функції n змінних є множина усіх можливих наборів довжини n або множина усіх вершин n-вимірного гіперкуба. Булеву функцію, яка визначена на усіх своїх наборах, називають ***повністю визначеною*** функцією. Прикладом такої функції є

Табл. 3.2. Неповністю визначена функція f_1 від аргументів x_1,x_2,x_3 та її довизначені варіанти f_2, f_3

x_1, x_2, x_3	f_1	f_2	f_3
0 0 0	0	0	0
0 0 1	1	1	1
0 1 0	–	0	1
0 1 1	0	0	0
1 0 0	1	1	1
1 0 1	–	0	1
1 1 0	1	1	1
1 1 1	–	0	1

функція, задана табл. 3.1.

Булева функція n змінних називається ***неповністю визначеною***, якщо вона визначена не на усіх своїх наборах. Наприклад, у табл. 3.2 показана не скрізь визначена функція f_1, невизначені набори якої помічені мінусом. На практиці, неповністю визначена функція передбачає, що набори, на яких вона не визначена, ніколи не трапляться при її застосуванні або вони не є вирішальними для подальшого сприйняття результату. Наприклад, десяткові цифри кодуються десятьма відповідними чотирибітовими наборами, а решта шість наборів — не використовуються. Також стани керуючого автомату кодуються наборами, яких може бути значно менше за 2^n, і такий автомат не може попадати у стан, який закодований невизначеним набором.

Неповністю визначена булева функція не підпадає під визначення булевої функції, яке приведене вище. Але при довільному довизначенні, як наприклад, у функціях f_2, f_3 у табл. 3.2, ця невідповідність знімається. Причому, якщо не визначено m наборів, то варіантів довизначення може бути 2^m.

Як показано вище, довільна булева функція на кожному з 2^n наборів може приймати значення 0 чи 1. Отже, можна побудувати до 2^{2^n} різних булевих функцій від n змінних.

3.2.2. Булеві функції від одного та двох аргументів

Чотири булевих функції від однієї змінної x це — константа нуль, повторення x, інверсія x, тобто, x та константа одиниця. Розглянемо булеві функції від двох змінних, число яких $2^{2^2} = 16$ і які приведені у табл. 3.3. Там само показані символи найбільш вживаних функцій. Серед них найвідоміші наступні:

$f_1 = x_1 \,\&\, x_2 = x_1 \wedge x_2 = x_1 \cdot x_2$ — кон'юнкція, яку називають логічним добутком, функцією "І";

$f_6 = x_1 \oplus x_2$ — сума за модулем два, функція "Виключне Або";

$f_7 = x_1 \vee x_2$ — диз'юнкція, логічне додавання, функція "Або";

$f_8 = x_1 \downarrow x_2$ — стрілка Пірса, функція "Або-Ні";

$f_9 = x_1 \sim x_2 = x_1 \leftrightarrow x_2$ — еквівалентність, функція "Інверсне Виключне Або";

$f_{13} = x_1 \rightarrow x_2$ — імплікація;

$f_{14} = x_1 / x_2$ — штрих Шефера, функція "І-Ні".

Табл. 3.3. Булеві функції від двох змінних

x_1, x_2	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
	0	&		x_1		x_2	$\oplus$	$\vee$	$\downarrow$	$\sim$	$\bar{x}_2$		$\bar{x}_1$	$\rightarrow$	/	1
0 0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0 1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1 0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1 1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

3.2.3. Суперпозиція булевих функцій

На основі розглянутих двомісних булевих функцій можна будувати нові складні функції за допомогою операції суперпозиції. Фактично ***операція суперпозиції*** полягає у підстановці інших булевих функцій замість аргументів. Наприклад, маючи функції $f_1(x_1, x_2)$, $f_2 = x_3$ та $f_3(x_4, x_5)$, одержуємо нову функцію $f_1(x_3, f_3(x_4, x_5))$.

Суперпозиція булевих функцій представляється у вигляді логічних формул. При такому представленні слід зважати на наступне:

— одна й та сама функція може бути представлена різними формулами;

— кожній формулі відповідає своя суперпозиція;

— кожна формула має певний параметр складності Θ, який відображає складність її обчислення або апаратної реалізації.

Очевидно, що серед формул, які реалізують одну і ту саму функцію, є така, яка має найменшу складність Θ. Значення функції складності Θ буде розглянуте у підрозділі присвяченому синтезу. Суть синтезу алгоритму чи обчислювальної схеми, які виконують задану булеву функцію, полягає у побудові ряду формул її обчислення та у виборі такої формули, яка має найменшу складність Θ. Така побудова найчастіше полягає у знаходженні початкової формули і у виконанні послідовності її еквівалентних перетворень, завдяки яким вона спрощується з одержанням найменшої складності Θ.

3.2.4. Булева алгебра перемикальних функцій

Як було зазначено вище, ***булева алгебра*** — це структура

$$\langle \{0,1\}; \cdot, \vee, \bar{} \rangle,$$

де визначені одна одномісна функція "¯" інверсії та дві двомісні операції кон'юнкції та диз'юнкції, які позначаються як "&" чи "·" та "∨", відповідно. У цій алгебрі справедливі три аксіоми:

A1: $x \vee y = y \vee x,\ x \cdot y = y \cdot x$ — комутативність;

A2: $x \vee (y \vee z) = (x \vee y) \vee z,\ \ x \cdot (y \cdot z) = (x \cdot y) \cdot z$ — асоціативність;

A3: $x \cdot (y \vee z) = (x \cdot y) \vee (x \cdot z),\quad x \vee (y \cdot z) = (x \vee y) \cdot (x \vee z)$ — дистрибутивність.

Слід взяти до уваги, що за традиційною домовленістю серед трьох операцій булевої алгебри інверсія має найвищий пріоритет, а кон'юнкція (логічне множення) менший пріоритет за інверсію, але більший пріоритет за диз'юнкцію.

Перетворення формул булевих функцій з використанням лише вказаних аксіом є малоефективним. Наступні співвідношення грають роль теорем і дають змогу спростити формулу краще.

Т1: $\overline{x \cdot y} = \bar{x} \vee \bar{y}$, $\overline{x \vee y} = \bar{x} \cdot \bar{y}$ — теореми де Моргана.

Т2: $x \vee x \cdot y = x$, $x \cdot (x \vee y) = x$ — закони поглинання.

Т3: $x \vee x = x$, $x \cdot x = x$;

Т4: $x \vee \bar{x} \cdot y = x \vee y$, — закон склеювання.

Т5: $x \cdot y \vee \bar{x} \cdot y = x$, $(x \vee y) \cdot (x \vee \bar{y}) = x$ — закони склеювання.

Т6: $x \vee \bar{x} = 1$, $x \cdot \bar{x} = 0$ — властивості доповнення.

Т7: $\bar{\bar{x}} = x$ — інволютивність, подвійне заперечення.

Т8: $x \vee 1 = 1$, $x \vee 0 = x$, $x \cdot 1 = x$, $x \cdot 0 = 0$ — властивості нуля та одиниці.

Приклад.

Спростити булеву функцію $F = \overline{\overline{x \cdot \bar{y} \vee \bar{y} \cdot x} \cdot x \vee (x \cdot \bar{y} \vee \bar{y} \cdot x) \cdot x}$.

Після застосування теореми де Моргана Т1 одержимо:

$F = \overline{\overline{x \cdot \bar{y} \vee \bar{y} \cdot x} \cdot x} \cdot \overline{(x \cdot \bar{y} \vee \bar{y} \cdot x) \cdot x}$.

Знову застосувавши теорему Т1 маємо:

$F = \left(\overline{\overline{(x \cdot \bar{y} \vee \bar{y} \cdot x)} \vee \cdot \bar{x}} \right) \cdot \left(\overline{(x \cdot \bar{y} \vee \bar{y} \cdot x)} \vee \cdot \bar{x} \right)$.

Застосувавши властивість інволютивності до перших дужок і теорему Т1 для других дужок, одержимо

$F = (x \cdot \bar{y} \vee \bar{x} \cdot y \vee \bar{x}) \cdot \left(\overline{x \cdot \bar{y}} \cdot \overline{\bar{y} \cdot x} \vee \bar{x} \right)$.

Послідовно застосувавши відношення Т2, Т4, а також Т1, одержується

$$F = (\overline{y} \vee \overline{x}) \cdot \left((\overline{x} \vee y) \cdot (x \vee \overline{y}) \vee \overline{x} \right).$$

Після розкриття внутрішніх дужок за правилами А3, Т6 та спрощення маємо

$$F = (\overline{y} \vee \overline{x}) \cdot (\overline{x} \cdot \overline{y} \vee x \cdot y \vee \overline{x}).$$

І застосовуючи правила Т2, Т4, Т5, Т6, А3, остаточний результат буде

$$F = (\overline{y} \vee \overline{x}) \cdot (\overline{x} \vee y) = \overline{x} \cdot \overline{y} \vee \overline{x} \cdot y = \overline{x}.$$

3.2.5. Логічні функції редукції

З комутативності, асоціативності диз'юнкції слідує, що диз'юнкція кількох змінних може бути виражена співвідношенням

$$f(x_1, x_2, \ldots, x_n) = x_1 \vee x_2 \vee \ldots \vee x_n = \bigvee_{i=1}^{n} x_i .$$

Отже, тут функція $f(x_1, x_2, \ldots, x_n)$, послідовно виконуючи диз'юнкцію над n змінними, дає один результат, тобто, вона редукує n змінних до однієї. Тому таку багатомісну функцію називають ***функцією редукції***. Аналогічно маємо функції редукції для кон'юнкції і суми за модулем два

$$x_1 \cdot x_2 \cdot \ldots \cdot x_n = \bigwedge_{i=1}^{n} x_i .$$

$$x_1 \oplus x_2 \oplus \ldots \oplus x_n = \sum_{i=1}^{n} x_i .$$

Теореми де Моргана поширюються для кількох змінних у виді:

$$\overline{x_1 \vee x_2 \vee \ldots \vee x_n} = \overline{x_1} \cdot \overline{x_2} \cdot \ldots \cdot \overline{x_n} = \bigwedge_{i=1}^{n} \overline{x_i} .$$

$$\overline{x_1 \cdot x_2 \cdot \ldots \cdot x_n} = \overline{x_1} \vee \overline{x_2} \vee \ldots \vee \overline{x_n} = \bigvee_{i=1}^{n} \overline{x_i} .$$

3.3. Реалізація булевих функцій у комп'ютерах

У багатьох мовах програмування визначений булевий тип даних `Boolean`, який має елементи `true` і `false`. Наприклад, у програмах на більшості алгоритмічних мов будь-яка операція порівняння повертає значення типу `Boolean`.

Крім того, у такій мові, як C, роль елемента `false` грає ціле число 0, а роль `true` — будь-яке ціле ненульове число. Наприклад, функцію $f = a \cdot b$ програмують мовою C наступними способами.

— оператором `if-else`:

```
if (~a) f = 0;
     else if (~b) f = 0;
          else f = 1;
```

— за допомогою логічної операції "І":

```
if (a && b) f = 1;
else f = 0;
```

— за допомогою тернарної операції

```
f = (a && b)? 1: 0;
```

Більш складну функцію, таку, яка задається таблицею істинності, можна реалізувати за допомогою оператора `switch-case`:

```
switch(s){
…
case i: f = 1; break;
…
 default: f = 0;
}
```

Тут `s` < 2^n — ціле число, яке представляє собою номер набору таблиці істинності, у рядку `case` *i* виконуються дії, які відповідають *i*-му набору цієї таблиці, а у рядку `default` — дії у решті випадків.

Часто довільну логічну функцію реалізують як масив булевих даних довжиною 2^n, *i*-та комірка якого відповідає *i*-му рядку таблиці істинності.

Інтегральні схеми (ІС) є як найважливішою продукцією промисловості електроніки, так і є основою усіх виробів комп'ютерної техніки. Транзистор — це елементарний компонент ІС. Найпростіший елемент ІС — інвертор — складається з двох комплементарних транзисторів зі структурою: метал-оксид-напівпровідник (КМОН), як показано на функціональній схемі на Рис. 3.3.

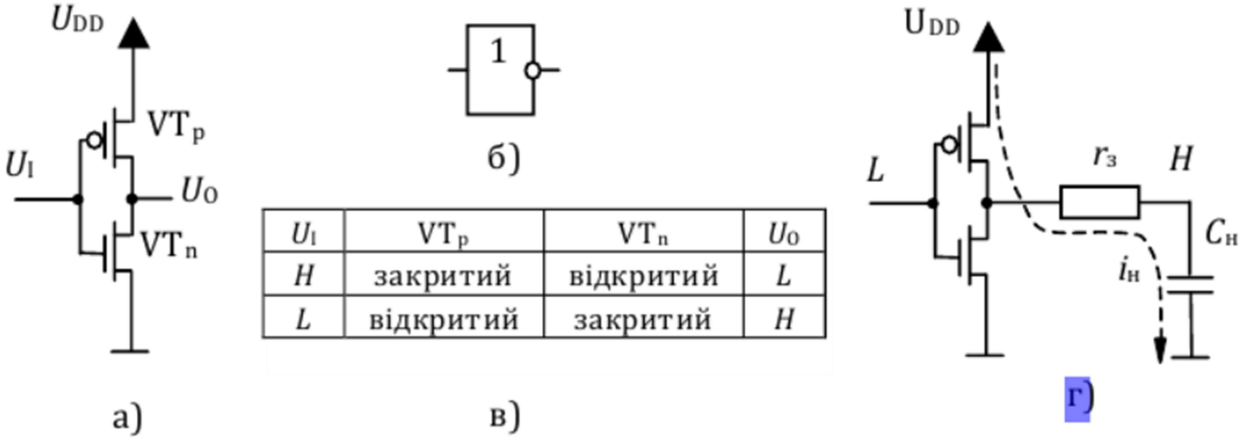
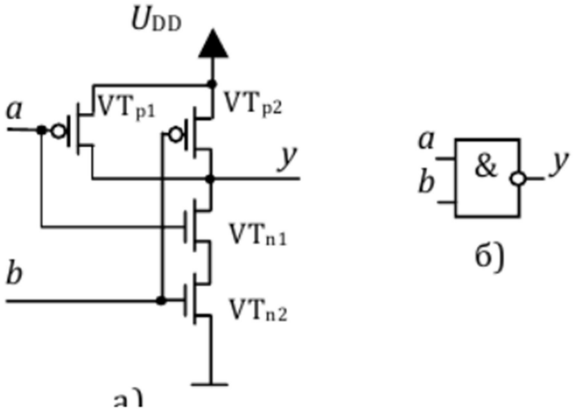


Рис. 3.3. Електрична схема інвертора (а), її умовне графічне зображення (б), таблиця режимів роботи (в) і пояснення дії (г)

p-канальний транзистор VTp у цьому інверторі завжди знаходиться у протилежному, тобто комплементарному стані відносно n-канального транзистора VTn, як показано у таблиці істинності на Рис. 3.3, в. У цій таблиці символ H (high) означає високий рівень сигналу, тобто логічну 1, а L (low) — низький, чи логічний 0.

Один 2-входовий логічний елемент (ЛЕ), названий логічним вентилем, може бути сформований з чотирьох транзисторів, як на Рис. 3.4. Якщо на обох входах a і b встановлений високий рівень, то транзистори VT_{n1} і VT_{n2} відкриваються, так що вихід y встановлюється у низький рівень, в той час, коли транзистори VT_{p1} і VT_{p2} є закритими. При іншій комбінації вхідних сигналів принаймні один з транзисторів VT_{p1}, VT_{p2} відкриється, а один з VT_{n1}, VT_{n2} — закриється і на виході буде високий рівень сигналу.



Входи		Вихід
a	*b*	*y*
L	*L*	*H*
L	*H*	*H*
H	*L*	*H*
H	*H*	*L*

в)

Рис. 3.4 Електрична схема вентиля І-Ні (а), його умовне графічне зображення (б), таблиця режимів роботи (в)

3.4. Аналітичне зображення булевих функцій

3.4.1. Канонічні форми булевої функції

Аналітична форма представлення булевої функції основана на суперпозиції кількох елементарних функцій. Вище було вказано, що таких суперпозицій може бути велика множина на основі багатьох видів елементарних функцій. Важливим є використання формального конструктивного підходу для формування такого представлення. Таким підходом є формування кананонічних форм представлення довільної логічної функції на підставі її таблиці істинності. Канонічна форма — це така форма, що однозначно репрезентує об'єкт, в даному випадку — логічну функцію. Але перед викладанням цього підходу слід дати ряд визначень.

Конституентою одиниці або ***мінтермом*** (англійською — minterm) є функція f (x_1, x_2,… , x_n), яка приймає значення 1 тільки на єдиному наборі аргументів. Така функція записується як кон'юнкція n різних булевих аргументів, частина яких можуть бути з інверсіями. Наприклад, 3-й рядок таблиці 3.1 кодується конституентою $\overline{x_1} \cdot \overline{x_2} \cdot x_3 \cdot x_4$, яка приймає одиничне значення лише на третьому наборі даної функції.

Відповідно, на решті наборів ця конституента дорівнює нулю. Власне, мінтерми часто кодуються номерами відповідних наборів.

Неважко збагнути, що будь-яку логічну функцію, яка задана таблицею істинності, можна представити як диз'юнкція конституент одиниці, які відповідають тим рядкам таблиці, у яких функція дорівнює одиниці, тобто:

$$f(x_1, x_2, \dots, x_n) = \bigvee_{i=1}^{n} f(\alpha_1,\alpha_2,\dots,\alpha_n) \cdot x_1^{\alpha_1}, x_2^{\alpha_2}, \dots, x_n^{\alpha_n},$$

де $\alpha_i \in \{0,1\}$ та $x_i^{\alpha_i} = \begin{cases} x_I & \text{при } \alpha_i = 1, \\ \bar{x}_I & \text{при } \alpha_i = 0. \end{cases}$

Ця канонічна форма називається ***досконалою диз'юнктивною нормальною формою*** (ДДНФ).

Приклад. Записати функцію $f_2 = f$ (1,4,6), яка задана табл. 3.2, у вигляді ДДНФ.

$$f(x_1, x_2, x_3) = 0 \cdot \bar{x}_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \vee 1 \cdot \bar{x}_1 \cdot \bar{x}_2 \cdot x_3 \vee 0 \cdot \bar{x}_1 \cdot x_2 \cdot \bar{x}_3 \vee 0 \cdot \bar{x}_1 \cdot x_2 \cdot x_3 \vee$$

$$\vee 1 \cdot x_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \vee 0 \cdot x_1 \cdot \bar{x}_2 \cdot x_3 \vee 1 \cdot x_1 \cdot x_2 \cdot \bar{x}_3 \vee 0 \cdot x_1 \cdot x_2 \cdot x_3 =$$

$$= \bar{x}_1 \cdot \bar{x}_2 \cdot x_3 \vee x_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \vee x_1 \cdot x_2 \cdot \bar{x}_3.$$

Іншою канонічною формою завдання логічної функції є досконала кон'юктивна нормальна форма (ДКНФ). Вона основана на ***конституенті нуля*** або ***макстермі*** (англійською — maxterm) — функції, яка приймає значення 0 на єдиному наборі. Конституента нуля записується як елементарна диз'юнкція усіх змінних. Наприклад, набору 0100 змінних x_1, x_2, x_3, x_4 відповідає конституента нуля $x_1 \vee \bar{x}_2 \vee x_3 \vee x_4$. Дійсно, при підстановці цього набору у змінні результат буде нуль, а на інших наборах — одиниця.

Досконала кон'юктивна нормальна форма представляється як

кон'юнкція конституент нуля, які відповідають нульовим булевим наборам функції.

Приклад. Записати функцію $f_2 = f\ (1,4,6)$, яка задана табл. 3.2, у вигляді ДКНФ.

$$f\,(x_1, x_2, x_3) = (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)\cdot(\bar{x}_1 \vee x_2 \vee \bar{x}_3)\cdot(x_1 \vee \bar{x}_2 \vee \bar{x}_3)\cdot(x_1 \vee \bar{x}_2 \vee x_3)\cdot(x_1 \vee x_2 \vee x_3).$$

3.4.2. Розкладання Шенона

ДДНФ є крайнім випадком розкладання Шенона. ***Розкладання Шенона*** — це формула, яка виконує перехід від представлення функції від n змінних до представлення через функції від n–1 змінної:

$$f\,(x_1, x_2, \ldots, x_n) = x_1 \cdot f(1, x_2, \ldots, x_n) \vee \bar{x}_1 \cdot f(0, x_2, \ldots, x_n).$$

Відношення легко узагальнюється для довільного числа змінних, якщо функції у правій частині розкласти таким самим чином, наприклад,

$$f\,(x_1, x_2, x_3) = x_1\cdot(x_2\cdot f(1, 1, x_3)) \vee x_1\cdot(\bar{x}_2\cdot f(1,0, x_3)) \vee$$

$$\vee\ \bar{x}_1\cdot(x_2\cdot f(0, 1, x_3)) \vee \bar{x}_1\cdot(\bar{x}_2\cdot f(0, 0, x_3)) =$$

$$= x_1\cdot x_2\cdot f(1, 1, x_3) \vee x_1\cdot \bar{x}_2\cdot f(1, 0, x_3) \vee \bar{x}_1\cdot x_2\cdot f(0, 1, x_3) \vee \bar{x}_1\cdot \bar{x}_2\cdot f(0, 0, x_3).$$

Якщо далі продовжити розкладання, то одержимо ДДНФ.

3.4.3. Алгебра Жегалкіна

У багатьох застосуваннях, наприклад, у системах контролю даних, завадостійкого кодування, часто використовується операція суми за модулем два. Для оперування з логічними функціями у цьому випадку доцільно використовувати алгебру Жегалкіна.

Алгебра Жегалкіна — це структура $\langle\{0,1\};\ \cdot\ , \oplus\rangle$, операціями якої є кон'юнкція та сума за модулем два. На неї розповсюджуються основні аксіоми:

A1: $x \oplus y = y \oplus x$, $x \cdot y = y \cdot x$, — комутативність;

A2: $x \oplus (y \oplus z) = (x \oplus y) \oplus z$, $x \cdot (y \cdot z) = (x \cdot y) \cdot z$; — асоціативність;

A3: $x \cdot (y \oplus z) = (x \cdot y) \oplus (x \cdot z)$; — дистрибутивність.

Для спрощення формул можуть бути використані наступні рівності:

$x \oplus x = 0$, $x \oplus 0 = x$, $x \oplus 1 = \bar{x}$,

$$x \oplus y = x \cdot \bar{y} \vee \bar{x} \cdot y = (x \vee y) \cdot (\bar{x} \vee \bar{y}) = (x \vee y) \cdot \overline{x \cdot y}\ .$$

Бачимо, що функція інверсії тут замінюється функцією суми за модулем два з константою 1. За своїм пріоритетом, операція $\oplus$ є найнижчою серед розглянутих логічних операцій інверсії, кон'юнкції та диз'юнкції.

Згадаємо, що ДДНФ складається з конституент одиниці, тобто, у будь-якому випадку лише одна конституента з усіх наявних дорівнює 1. Сума за модулем 2 єдиної одиниці і кількох нулів також дорівнює одиниці. Тому ДДНФ можна замінити сумою за модулем два тих самих конституент одиниці.

Приклад. Знайти для функції f (1,4,6) представлення у алгебрі Жегалкіна.

$$f(1,4,6) = \bar{x}_1 \cdot \bar{x}_2 \cdot x_3 \oplus x_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \oplus x_1 \cdot x_2 \cdot \bar{x}_3.$$

Таке представлення називається ***досконалою поліноміальною нормальною формою*** (ДПНФ).

Можна піти далі, взявши до уваги, що $\bar{x} = 1 \oplus x$ і замінивши усі інверсії в конституентах.

Приклад. Для прикладу ДПНФ, який показано вище, застосувати заміну інверсії. Одержимо:

$$f(1,4,6) = (1 \oplus x_1) \cdot (1 \oplus x_2) \cdot x_3 \oplus x_1 \cdot (1 \oplus x_2) \cdot (1 \oplus x_3) \oplus x_1 \cdot x_2 \cdot (1 \oplus x_3) =$$

$$= (1 \cdot 1 \oplus 1 \cdot x_1 \oplus 1 \cdot x_2 \oplus x_1 \cdot x_2) \cdot x_3 \oplus x_1 \cdot (1 \cdot 1 \oplus 1 \cdot x_2 \oplus 1 \cdot x_3 \oplus x_2 \cdot x_3) \oplus 1 \cdot x_1 \cdot x_2 \oplus x_1 \cdot x_2 \cdot x_3 =$$

$= x_3 \oplus x_1 \oplus x_1 \oplus x_2 \cdot x_3 \oplus x_1 \cdot x_2 \oplus x_1 \cdot x_2 \oplus x_1 \cdot x_3 \oplus x_1 \cdot x_2 \cdot x_3 \oplus x_1 \cdot x_2 \cdot x_3 \oplus x_1 \cdot x_2 \cdot x_3 =$

$= x_3 \oplus 0 \oplus x_2 \cdot x_3 \oplus 0 \oplus x_1 \cdot x_3 \oplus 0 \oplus x_1 \cdot x_2 \cdot x_3 = x_3 \oplus x_2 \cdot x_3 \oplus x_1 \cdot x_3 \oplus x_1 \cdot x_2 \cdot x_3$.

Результат такого перетворення називається ***канонічним поліномом Жегалкіна***.

3.4.4. Функціонально повні системи булевих функцій

Вище було показано, що довільну булеву функцію можна представити у ДДНФ чи ДКНФ. Отже, структура, що складається з кон'юнкції, диз'юнкції та інверсії придатна для аналітичного представлення булевої функції будь-якої складності. Також представлення у ДПНФ свідчить про те, що такі самі властивості має структура, в яку входять диз'юнкція та сума за модулем два. Такі структури називаються ***функціонально повними***.

При проектуванні цифрових пристроїв спочатку вибирають елементний базис, який реалізує ті чи інші елементарні булеві функції. Цей базис найчастіше залежить від технології виготовлення цих пристроїв. Далі створюються досконалі форми булевих функцій у відповідності з алгоритмом роботи пристрою. Така форма приводиться еквівалентними перетвореннями у форму, яка може бути реалізована у даному елементному базисі з мінімізованими апаратними витратами та гідною швидкодією. У зв'язку з цим постає питання, який можливий мінімальний набір функцій такого елементного базису. Це питання вирішується вибором функціонально повної системи булевих функцій.

Функціонально повний клас булевих функцій P_2 повинен мати властивість замкненості, тобто, будь-який результат суперпозиції таких функцій повинен належати цьому класу. У зв'язку з цим, серед функціонально замкнених класів функцій виділяють ***передповний клас функцій*** або клас Поста. Цей клас характерний тим, що якщо таку функцію доповнити довільною функцією алгебри логіки, то утвориться

повний клас P_2. Звідси слідує, що клас P_2 повинен вміщувати такі функції, які не вміщуються повністю у жодному передповному класі.

Було знайдено п'ять передповних класів:

1) клас функцій, що зберігають константу 0:

$$T_0 = \{f: f(0, 0, \ldots, 0) = 0\};$$

2) клас функцій, що зберігають константу 1:

$$T_1 = \{f: f(1, 1, \ldots, 1) = 1\};$$

3) клас самодвоїстих функцій:

$$S = \{ f: f(\bar{x}_1,\bar{x}_2,\ldots,\bar{x}_n) = \overline{f(x_1, x_2,\ldots, x_n)} \};$$

4) клас монотоних функцій:

$$M = \{f: \forall i(a_i \le b_i) \Rightarrow f(a_1, a_2,\ldots, a_n) \le f(b_1, b_2,\ldots, b_n);$$

5) клас лінійних функцій:

$$L = \{f: f(x_1, x_2,\ldots, x_n) = a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus,\ldots,\oplus a_n x_n,\ a_i \in \{0, 1\}\}.$$

Прикладами булевих функцій, які зберігають константу 0, є функції $f_0,\ldots f_7$ з таблиці 3.3., а функцій, які зберігають 1 — функції з непарними номерами f_1, f_3, $\ldots f_{15}$ з тієї самої таблиці. Самодвоїстими функціями є функції f_3, f_5, f_{10}, f_{12} з таблиці 3.3, тому що на протилежних наборах вони мають протилежні значення. Лінійними є функції f_0, f_3, f_5, f_6, f_9, f_{10}, f_{12}, f_{15} (табл..3.3), бо

$$f_0 = 0, f_3 = x_1, f_5 = x_2, f_6 = x_1 \oplus x_2, f_9 = 1 \oplus x_1 \oplus x_2, f_{10} = 1 \oplus x_2, f_{12} = 1 \oplus x_1, f_{15} = 1.$$

Монотонними вважаються функції, у яких є відношення порядку, тобто, функція приймає більше значення для аргументів b_i, які є більшими, тобто, для них $\forall i(a_i \le b_i)$. Так, функції f_0, f_1, f_3, f_5, f_7, f_{15} є монотонними (табл. 3.3). Але наприклад, f_2 не є монотонною, оскільки $f_2(1,0) \not\le f_2(1,1)$ при тому, що $(1,0) \le (1,1)$.

Була доведена теорема: система булевих функцій є повною, якщо вона вміщує принаймні по одній функції, як не належить до одного з п'яти предповних класів, тобто, хоча б одну функцію, що не зберігає константу 0, одну функцію, що не зберігає 1, одну несамодвоїсту функцію, одну нелінійну функцію та одну немонотонну функцію.

У табл. 3.4. показано плюсом, до якого передповного класу належить та чи інша булева функція, а мінусом — до якого не належить.

Табл. 3.4. Належність булевих функцій до перед повних класів

Передповний клас	f_0 0	f_1 $x_1 \cdot x_2$	f_3 x_1	f_6 $x_1 \oplus x_2$	f_7 $x_1 \vee x_2$	f_8 $x_1 \downarrow x_2$	f_9 $x_1 \sim x_2$	f_{12} $\bar{x}_1$	f_{13} $x_1 \to x_2$	f_{14} x_1 / x_2	f_{15} 1
T_0	+	+	+	+	+	–	–	–	+	–	–
T_1	–	+	+	–	+	–	+	–	–	–	+
S	–	–	+	–	–	–	–	+	–	–	–
M	+	+	+	–	+	–	–	–	–	–	+
L	+	–	+	+	–	–	+	+	–	–	+

Отже, для формування функціонально повного класу булевих функцій слід вибрати з таблиці 3.4 такі стовпчики, об'єднання яких дає змогу зібрати знаки мінус у всіх п'яти позиціях передповних класів. Цей процес ілюструється Рис. 3.5. Можна бачити, що є стовпчики повністю відмічені мінусом, тобто, є класи, які складаються лише з однієї функції. Це $P_2 = \{\downarrow\}$ та $P_2 = \{/\}$. Отже, маючи лише функцію Штрих Шефера, тобто І-Ні, можна за допомогою суперпозиції виразити будь-яку булеву функцію довільної складності.

Повні класи, які мають лише по дві функції, це $\{\cdot,\neg\}$, $\{\vee,\neg\}$, $\{\to,1\}$, $\{\to,\neg\}$, $\{\to,\sim\}$. У практиці проектування цифрових автоматів знаходять застосування також такі повні класи з трьома функціями, як $\{\cdot,\vee,\neg\}$, $\{\cdot,\oplus,\neg\}$, $\{\cdot,\oplus,1\}$.

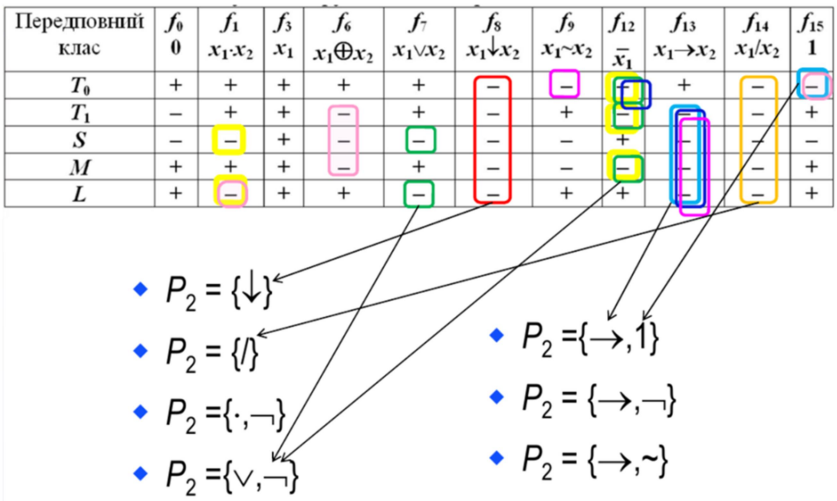


Рис.3.5. Формування функціонально повних класів булевих функцій

Реалізація констант 0 та 1 потребує мінімальних витрат. Тому для одержання повного класу досить вибрати нелінійні та немонотонні функції і додати до них ці константи. Так, мають застосування класи $\{\downarrow,0,1\}$, $\{\to,0,1\}$, $\{\cdot,\oplus,0,1\}$, $\{\vee,\oplus,0,1\}$.

3.4.5. Принцип двоїстості булевих функцій

Логічну функцію $f(x_1, x_2, \dots, x_n)$ називають ***двоїстою*** до функції

$$f^*(x_1, x_2, \dots, x_n) = \overline{f(\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)}\ .$$

При цьому виконується рівність

$$f(x_1, x_2, \dots, x_n) = (f^*(x_1, x_2, \dots, x_n))^*.$$

Принцип двоїстості полягає у наступному. Якщо формула $U = C(f_1,\dots,f_S)$ реалізує булеву функцію $f(x_1, x_2, \dots, x_n)$, то формула $U^* = C(f^*_1,\dots,f^*_S)$, яка одержана з U заміною функцій $f_1,\dots,f_S$ на двоїсті функції $f^*_1,\dots,f^*_S$, реалізує функцію $f^*(x_1, x_2, \dots, x_n)$, яка є двоїстою до функції $f(x_1, x_2, \dots, x_n)$. Формулу U^* називають двоїстою до формули U.

Якщо маємо клас функцій $\{\cdot,\vee,\neg,0,1\}$, то принцип двоїстості застосовують наступним чином. У формулі U всюди замінюють 0 на 1, 1 на 0, $\cdot$ на $\vee$, $\vee$ на $\cdot$, результатом чого буде двоїста формула U^*.

Наприклад, є відоме відношення де Моргана: $\overline{x \cdot y} = \bar{x} \vee \bar{y}$. Якщо

застосувати принцип двоїстості, то одержимо відношення: $\overline{x \vee y} = \bar{x} \cdot \bar{y}$.

3.4.6. Завдання

1. Спростити вираз:

1) $b\bar{c}\,(c \vee a\bar{c}) \vee (\bar{a} \vee \bar{c})(\bar{a}b \vee \bar{a}c)$.

2) $\overline{(a \vee b)}\,\overline{(a\,b\,c)}\,\overline{(\bar{a}\,c)}$.

3) $\overline{(ab \vee \bar{b}c) \vee (\bar{a}\,\bar{c} \vee a\,c)}$.

4) $\overline{(\bar{a}\,\bar{c} \vee b)} \vee b \vee a\,c$.

5) $(x \vee y)(\overline{z} \vee \overline{u}\,\overline{y})$. .

6) $\overline{x \vee \overline{y \vee \overline{z}}}$.

7) $(x\overline{y} \vee \overline{x}\,z)\,\overline{u \vee y\overline{z}}$.

2. Застосовуючи основні еквівалентності булевої алгебри (основні теореми), довести еквівалентність формул Ф і Ψ.

1) $\Phi = \overline{(x_3 / x_2)} / ((x_3 \downarrow x_1) \vee \overline{x_3})$, $\Psi = (x_1 \downarrow (x_2 \to x_1)) \to (x_2 \vee x_3)$.

2) $\Phi = x_3 / ((x_1 \oplus x_2) \sim x_2)$, $\Psi = \overline{(x_2 \downarrow x_3)} / ((x_1 \sim x_3) \downarrow x_1)$.

3) $\Phi = x_2x_3\,\overline{(x_3 / (x_3 \downarrow x_2))} \vee (x_2 \sim (x_3 \oplus x_1)))$, $\Psi = (x_2 / x_3) \downarrow (x_1x_3)$.

4) $\Phi = (x_2 / x_1) \sim (x_3 \oplus x_1) \sim (x_1 \vee x_3)$, $\Psi = \overline{x_1} \vee (x_2 \sim x_3)$.

3. Функцію $f(x_1, x_2, x_3, x_4) = 1$ на наборах 0, 1, 2, 5, 6, 7, 9, 10, 11, 15 представити як ДДНФ, ДКНФ і поліном Жегалкіна.

4. Перевірити належність функцій до класів T_0, T_1, L, S, M.

1) $z(x \sim y)$.

2) $x \vee (y / z)$.

3) $\overline{y} \oplus xz$.

4) $(x / y) \to z.$

5) $(xy) / z.$

5. Застосовуючи принцип двоїстості, спростити вирази.

1) $(\overline{x} \vee \overline{y}) \cdot (x \vee \overline{y})(y \vee z).$

2) $(\overline{x} \vee y) \cdot (z \vee y)(x \vee z).$

3) $x \cdot \overline{z} \vee \overline{x} \cdot y \vee \overline{x \cdot z}.$

4) $\overline{x \cdot \overline{y} \vee \overline{x}}.$

5) $(\overline{a} \vee \overline{b} \vee \overline{c}) \cdot (\overline{c} \vee \overline{d})(b \vee c)$

6. Побудувати таблицю істинності для логічної функції.

1) $f = x(x \vee \overline{y} \vee z).$

2) $f = \overline{y}\ (x \vee y \vee \overline{z}).$

3) $f = (x \vee y)(x \vee z).$

4) $f = (\overline{x} \vee \overline{y})(x \vee z)$

5) $f = \overline{x}\ \overline{y} \vee z(x \vee y)$

6) $f = xy(x \vee y \vee z).$

3.5. Мінімізація булевих функцій

3.5.1. Властивості ДНФ

Вище було показано, як можна мінімізувати булеву функцію, використовуючи при послідовних перетвореннях її формули властивості аксіом і теорем булевої алгебри. При цьому такі перетворення не обов'язково призводять до оптимального рішення та ґрунтуються на вправності виконавця таких перетворень. У даному підрозділі викладаються поширені методи формальної мінімізації булевих функцій, які гарантовано призводять для одержання ефективних рішень.

Елементарною кон'юнкцією називається кон'юнкція скінченного числа різних булевих змінних, які є інвертованими або ні.

Диз'юнктивною нормальною формою (ДНФ) називається диз'юнкція елементарних кон'юнкцій.

Мінімальною ДНФ називається ДНФ, яка вміщує найменшу кількість літер, які позначають операнди функції. Ця кількість характеризує собою просторову складність Θ функції. Наприклад, ця кількість відображає число входів логічних вентилів першої ланки логічної схеми і отже, апаратну складність пристрою, який реалізує дану функцію.

Булева функція $g(x_1, x_2, \dots, x_n)$ є ***імплікантою*** булевої функції $f(x_1, x_2, \dots, x_n)$, якщо для довільного набору змінних, на якому $g(x_1, x_2, \dots, x_n) = 1$, так само $f(x_1, x_2, \dots, x_n) = 1$.

Приклад. Для ДДНФ $f(x_1, x_2, x_3) = \bar{x}_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \vee \bar{x}_1 \cdot x_2 \cdot \bar{x}_3 \vee x_1 \cdot \bar{x}_2 \cdot x_3$ знайти імпліканти. Є сім імплікант:

$g_1 = \bar{x}_1 \cdot \bar{x}_2 \cdot \bar{x}_3$;

$g_2 = \bar{x}_1 \cdot x_2 \cdot \bar{x}_3$;

$g_3 = \bar{x}_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \vee \bar{x}_1 \cdot x_2 \cdot \bar{x}_3 = \bar{x}_1 \cdot \bar{x}_3 \cdot (\bar{x}_2 \vee x_2) = \bar{x}_1 \cdot \bar{x}_3$;

$g_4 = x_1 \cdot \bar{x}_2 \cdot x_3$;

$g_5 = \bar{x}_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \vee x_1 \cdot \bar{x}_2 \cdot x_3$;

$g_6 = \bar{x}_1 \cdot x_2 \cdot \bar{x}_3 \vee x_1 \cdot \bar{x}_2 \cdot x_3$

$g_7 = \bar{x}_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \vee \bar{x}_1 \cdot x_2 \cdot \bar{x}_3 \vee x_1 \cdot \bar{x}_2 \cdot x_3$.

Як бачимо з прикладу, у однієї функції бувають великі та малі за розміром імпліканти, причому деякі малі імпліканти входять у склад великих імплікант. ***Проста імпліканта*** g елементарної кон'юнкції f — це така імпліканта, що ніяка її частина не є імплікантою функції f. У даному прикладі імпліканти $g_3 = \bar{x}_1 \cdot \bar{x}_3$, $g_4 = x_1 \cdot \bar{x}_2 \cdot x_3$ є простими.

Неважко довести наступні **теореми**.

1. Диз'юнкція довільного числа імплікант булевої функції f також є імплікантою цієї функції.

2. Будь-яка булева функція дорівнює диз'юнкції усіх своїх простих імплікант. Така диз'юнкція простих імплікант називається ***скороченою ДНФ***.

Для наведеного прикладу скороченою ДНФ є

$$f = g_3 \vee g_4 = \bar{x}_1 \cdot \bar{x}_3 \vee x_1 \cdot \bar{x}_2 \cdot x_3.$$

Формування множини імплікант, виділення з них простих імплікант та складання скорочених ДНФ виконуються на першому етапі оптимізації булевої функції. Причому таких скорочених ДНФ для однієї функції може бути кілька і в них можуть траплятись надлишкові прості імпліканти. Причому, якщо видалити таку надлишкову імпліканту, то функція не зміниться. Скорочена ДНФ, у якій відсутні надлишкові імпліканти, називається ***тупиковою ДНФ***.

Можна довести, що мінімальною ДНФ є тупикова ДНФ. Слід мати на увазі, що для однієї складної булевої функції трапляється множина різних тупикових ДНФ так само, як і різних мінімальних ДНФ. Одержання

множини тупикових ДНФ та вибір з них оптимальної представляє собою останній етап мінімізації булевої функції.

3.5.2. Метод Квайна

Метод Квайна використовує дві теореми булевої алгебри: теорему склеювання:

$$A \cdot x \vee A \cdot \bar{x} = A;$$

та теорему поглинання

$$A \cdot x \vee A = A,$$

де A — довільний вираз. З відношення поглинання виходить, що довільна елементарна кон'юнкція поглинається її частиною. І навпаки — операцією ***розгортування***, яка є зворотною до склеювання, можна одержати довільну просту імпліканту:

$$A = A \cdot (x \vee \bar{x}) \cdot = A \cdot x \vee A \cdot \bar{x}.$$

Суть методу полягає у виконанні усіх можливих склеювань і потім усіх поглинань, результатом чого є скорочена ДНФ, яка є початковою формулою для оптимізації.

На першому етапі отримують скорочену ДНФ, користуючись поняттями суміжних минтермів та імплікант. Основною операцією спрощення є операція склеювання. ***Суміжними*** називаються мінтерми, що відрізняються формою входження в них лише одного аргументу. Наприклад, суміжними є мінтерми:

$$\bar{x}_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \text{ та } \bar{x}_1 \cdot x_2 \cdot \bar{x}_3, \ \bar{x}_1 \cdot x_2 \cdot \bar{x}_3 \text{ та } x_1 \cdot x_2 \cdot \bar{x}_3.$$

Кожні два суміжних мінтерми склеюються по аргументу, який розрізняється, що призводить до заміни їх однією кон'юнкцією з числом аргументів на одиницю менше.

Кон'юнкції, які одержуються в результаті склеювання двох суміжних мінтермів, є імплікантами. Різні імпліканти з однаковим числом аргументів, в свою чергу, можуть виявитися суміжними, що дозволяє виконувати склеювання їх між собою. Імпліканта покриває усі мінтерми, в результаті, склеювання яких вона отримана.

Імпліканти, що не склеюються з іншими, виявляються простими. Прості імпліканти та мінтерми, які не мають суміжних їм для склеювання, входять в результуючу ДНФ, тобто, у скорочену ДНФ.

На другому етапі мінімізації виконують усунення надлишкових імплікант зі скороченої ДНФ. Суть мінімізації булевої функції полягає в тому, що множину конституент одиниці, які формують ДДНФ, можна покрити різними множинами імплікант. Оптимальний результат буде визначатись мінімальним покриттям імплікантами. Згадаємо, що таке покриття, за Рис. 2.5.

Приклад. Мінімізувати функцію:

$$y = F(a,b,c) = a \cdot b \cdot c \vee \bar{a} \cdot b \cdot c \vee a \cdot \bar{b} \cdot c \vee a \cdot \bar{b} \cdot \bar{c}.$$

Зіставляючи по черзі всі мінтерми, можна встановити, що склеюються перший і другий, перший і третій, третій і четвертий мінтерми, так що після склеювання одержимо:

$$y = b \cdot c \vee a \cdot c \vee a \cdot \bar{b}.$$

Кожна з результуючих імплікант є простою, а функція — скороченою ДНФ (Рис. 3.6).

$$y = F(a,b,c) = a \cdot b \cdot c \vee \bar{a} \cdot b \cdot c \vee a \cdot \bar{b} \cdot c \vee a \cdot \bar{b} \cdot \bar{c}.$$

Рис. 3.6. Покриття імплікантами ДДНФ

Але покриття мінтермів трьома імплікантами — це надмірне рішення. Перша з них покриває перший і другий мінтерми, остання — третій і четвертий. Імпліканта, що покриває перший і третій мінтерми є надмірною.

Після усунення зі скороченої ДНФ надлишкової імпліканти одержуємо тупикову ДНФ. У прикладі тупикова і вона ж мінімальна ДНФ має вигляд:

$$y = b \cdot c \vee a \cdot \bar{b}.$$

Отже, на другому етапі для одержання мінімальна ДНФ необхідно зі скороченої ДНФ видалити усі прості імпліканти. Для цього зручно використовувати спеціальну таблицю-матрицю. Рядки такої матриці відмічаються простими імплікантами, а стовпці — конституентами одиниці, тобто, елементами скороченої ДНФ булевої функції. Відповідна клітинка цієї матриці на перетині рядка імпліканти та стовпця з конституенти, яка в ньому використовується, відмічається хрестиком чи кольором. На основі аналізу такої матриці вибираються імпліканти тупикової ДНФ.

Для пояснення такого аналізу та вибору розглянемо приклад. Цей приклад також детально пояснює хід методу мінімізації булевої функції, яка дана у довільному представленні.

Приклад. Мінімізувати булеву функцію *f* методом Квайна:

$$f(x_1, x_2, x_3) = \overline{(\bar{x}_1 \vee \bar{x}_3)\cdot(x_2 \vee \bar{x}_3)} \vee \bar{x}_1 \cdot \bar{x}_2 \vee x_1 \cdot (x_2 \vee \bar{x}_2 \cdot x_3).$$

На *першому етапі* отримують скорочену ДНФ. Для цього видаляються інверсії, дужки та поглинаються терми:

$$f = \overline{\bar{x}_1 \vee \bar{x}_3} \vee \overline{x_2 \vee \bar{x}_3} \vee \bar{x}_1 \cdot \bar{x}_2 \vee x_1 \cdot x_2 \vee x_1 \cdot \bar{x}_2 \cdot x_3 =$$

$$= x_1 \cdot x_3 \vee \bar{x}_2 \cdot x_3 \vee \bar{x}_1 \cdot \bar{x}_2 \vee x_1 \cdot x_2 \vee x_1 \cdot \bar{x}_2 \cdot x_3 = x_1 \cdot x_3 \vee \bar{x}_2 \cdot x_3 \vee \bar{x}_1 \cdot \bar{x}_2 \vee x_1 \cdot x_2.$$

За допомогою операції розгортування відновлюється ДДНФ:

$$f = x_1 \cdot (\bar{x}_2 \vee x_2) \cdot x_3 \vee (\bar{x}_1 \vee x_1) \cdot \bar{x}_2 \cdot x_3 \vee \bar{x}_1 \cdot \bar{x}_2 \cdot (\bar{x}_3 \vee x_3) \vee x_1 \cdot x_2 \cdot (\bar{x}_3 \vee x_3) =$$

$$= x_1 \cdot \bar{x}_2 \cdot x_3 \vee x_1 \cdot x_2 \cdot x_3 \vee \bar{x}_1 \cdot \bar{x}_2 \cdot x_3 \vee x_1 \cdot \bar{x}_2 \cdot x_3 \vee \bar{x}_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \vee \bar{x}_1 \cdot \bar{x}_2 \cdot x_3 \vee x_1 \cdot x_2 \cdot \bar{x}_3 \vee x_1 \cdot x_2 \cdot x_3.$$

Маємо ДДНФ та її конституенти одиниці:

$$f = x_1 \cdot \bar{x}_2 \cdot x_3 \vee x_1 \cdot x_2 \cdot x_3 \vee \bar{x}_1 \cdot \bar{x}_2 \cdot x_3 \vee \bar{x}_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \vee x_1 \cdot x_2 \cdot \bar{x}_3.$$

За допомогою різних операцій склеювання одержується скорочена ДНФ:

$$f = x_1 \cdot x_3 \vee \bar{x}_2 \cdot x_3 \vee \bar{x}_1 \cdot \bar{x}_2 \vee x_1 \cdot x_2.$$

На *другому етапі* виконується, власне, оптимізація скороченої ДНФ. Формується імплікантна матриця Квайна як у табл. 3.5. У ній спочатку відмічаються хрестиком ті комірки, які відповідають входженню конституенти одиниці у ту чи іншу мінімальну імпліканту.

Слід відмітити, що число N хрестиків у одному рядку завжди є ступенем 2. Більш того, n-містної функції це число дорівнює $N = 2^{n-k}$, де k — число елементів у простій імпліканті.

Таблиця 3.5. Матриця Квайна

Прості імпліканти	Конституенти одиниці функції f				
	$\bar{x}_1 \cdot \bar{x}_2 \cdot \bar{x}_3$	$\bar{x}_1 \cdot \bar{x}_2 \cdot x_3$	$x_1 \cdot \bar{x}_2 \cdot x_3$	$x_1 \cdot x_2 \cdot \bar{x}_3$	$x_1 \cdot x_2 \cdot x_3$
$x_1 \cdot x_3$			X		X
$\bar{x}_2 \cdot x_3$		X	X		
$\bar{x}_1 \cdot \bar{x}_2$	X	X			
$x_1 \cdot x_2$				X	X

Далі у матриці шукаються стовпці, які мають лише один хрестик. Відповідні цим хрестикам прості імпліканти називаються ***базисними***. В

табл. 3.5 такі хрестики обведені контуром. Вони складають ***ядро булевої функції.*** Ядро обов'язково входить у будь-яку тупикову ДНФ. Тут по одному хрестику мають перший та четвертий стовпці, отже, базисними імплікантами є $\bar{x}_1 \cdot \bar{x}_2$ та $x_1 \cdot x_2$.

Потім розглядаються різні варіанти вибору сукупності простих імплікант, які покривають хрестиками решту стовпців імплікантної матриці. Вибираються варіанти покриття з мінімальним сумарним числом літер у такій сукупності імплікант. Тут є лише два варіанти: до базисних імплікант додається імпліканта $x_1 \cdot x_3$ або імпліканта $\bar{x}_2 \cdot x_3$. Результатами є дві тупикові ДНФ.

$$f = \bar{x}_1 \cdot \bar{x}_2 \vee x_1 \cdot x_2 \vee x_1 \cdot x_3 = \bar{x}_1 \cdot \bar{x}_2 \vee x_1 \cdot x_2 \vee \bar{x}_2 \cdot x_3.$$

За критерієм складності Θ, обидві ДНФ мають однакову складність. Тому результатом оптимізації вибирається довільна тупикова ДНФ як мінімальна.

Аналогічно виконується мінімізація ДКНФ. При цьому шукаються прості імпліканти КНФ, які є кон'юнкціями, що одержані при склеюванні сусідніх макстермів.

3.5.3. Метод Вейча-Карно

Для полегшення пошуку тупикових ДНФ або КНФ при числі аргументів не більше 4 — 8 використовується метод Вейча-Карно. Він заснований на табличному поданні значень мінтермів у вигляді карт Карно або діаграм Вейча. Якщо проаналізувати графічне представлення булевої функції як, наприклад, на Рис. 3.1, то можна пересвідчитись, що кожні дві суміжні вершини гіперкубу, що представляють конституенти одиниці, які відрізняються у одній змінній — у однієї конституенти ця змінна з інверсією, а у іншої — без. Такі конституенти можна злити в одну імпліканту. Так само, чотири вершини гіперкубу, які формують одну

його грань, відрізняються між собою у двох змінних і також можуть бути злиті у одну імпліканту. Отже, графічне представлення булевої функції можливо застосувати для її мінімізації. Наприклад, функція, яка розглядалась у попередньому прикладі, кодується кубом, як на Рис. 3.4, а.

Пари мінтермів — конституент одиниці, які знаходяться на суміжних вершинах гіперкубу, склеюються у просту імпліканту, що відмічається (покривається) замкненою лінією — контуром. При цьому наочно видно, які мінтерми склеюються в імпліканти ядра — вони покриваються лише одним способом. Чотири мінтерми, які знаходяться на одній грані гіперкуба також склеюються у просту імпліканту. Так само вісім мінтермів, розташованих на гіперплощині також склеюються у просту імпліканту.

Але представляти та аналізувати гіперкуб розмірністю більше трьох є складним завданням. Щоб полегшити це, з гіперкуба роблять його розгортку на площину. Наприклад, куб, зображений на Рис. 3.7, а, можна розрізати на дві частини площиною, яка проходить через його центр вертикально і розгорнути його на площину так, як показано на Рис. 3.7, б. Наступним кроком такої трансформації є заміна розгортки так званою діаграмою Вейча (синонім — карта Карно).

Карта Карно (англійською — Karnaugh map) представляє n-вимірний гіперкуб і має 2^n клітин. Вона має квадратну або прямокутну форму. На Рис. 3.6, в показана карта Карно порядку n=3, яка трансформується куба на Рис. 3.6, а через розгортку на Рис. 3.6, б. Аналогічно будуються карти Карно порядку 4 і 5, які показані на Рис. 3.8, а та 3.8, б.

Клітини рядків і стовпців карти Карно, що відповідають змінній, що дорівнює 1 у мінтермах, відзначаються товстою рискою. У клітинах, які не виділені рискою, ця змінна дорівнює 0. У кожній клітинці карти на Рис. 3.8 розміщено номер відповідного мінтерму, у даному випадку — шістнадцятковий номер для зручності його асоціації з мінтермом.

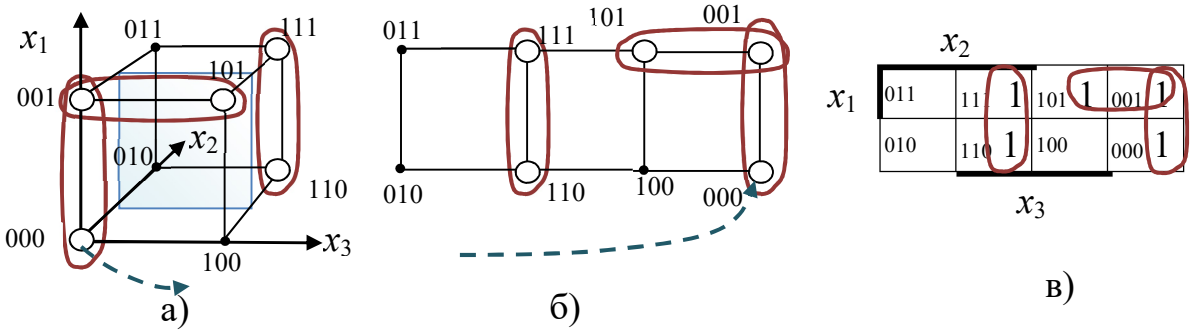


Рис. 3.7. Поадання булевої функції геометричним способом (а), розгортка куба (б) та відповідна карта Карно (в)

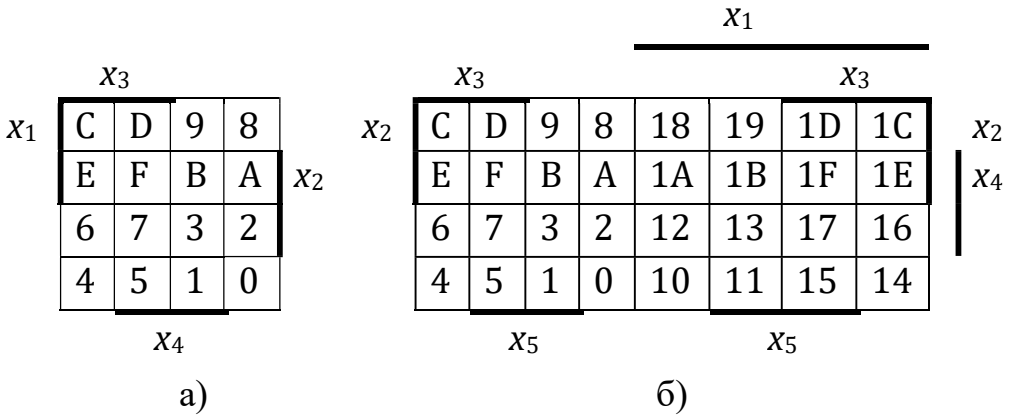


Рис. 3.8. Карти Карно для функцій від 4 і 5 змінних.

У клітинки карти заноситься значення 1 мінтермів ДДНФ, причому суміжні мінтерми розташовуються в сусідніх клітинах. Сусідніми вважаються клітини, що мають спільні сторони, а також ті, що розташовані на краях одних і тих самих рядків і стовпців карти. При цьому слід зважати на те, що суміжні вершини гіперкубу при його розгортці можуть опинитись на краях розгортки, як наприклад, ті, що позначені 001 і 011 на Рис. 3.7.

Сусідні клітини, що містять 1, включають в контур (незамкнений, якщо мінтерми знаходяться в крайніх клітинах карти). Такий контур означає склеювання пари сусідніх мінтермів по одному аргументу. Склеюванню по двох аргументах відповідають контури, що охоплюють чотири сусідніх клітини, які розташовані у вигляді квадрата або

прямокутника. Так само, якщо пари клітин розташовані на протилежних краях карти, вони охоплюються незамкненим контуром (контур розривається через розгортування гіперкуба на площину).

Перевірка того, що склеювання відбулось правильно, є те, що у контур попадають два мінтерми, які розрізняються за одною змінною або чотири мінтерми, які розрізняються за двома змінними і т. д. Типовою помилкою є склеювання не 2, 4, 8 клітинок, а їх іншого числа. На Рис. 3.9 показані приклади склеювання мінтермів на карті Карно для чотирьох змінних.

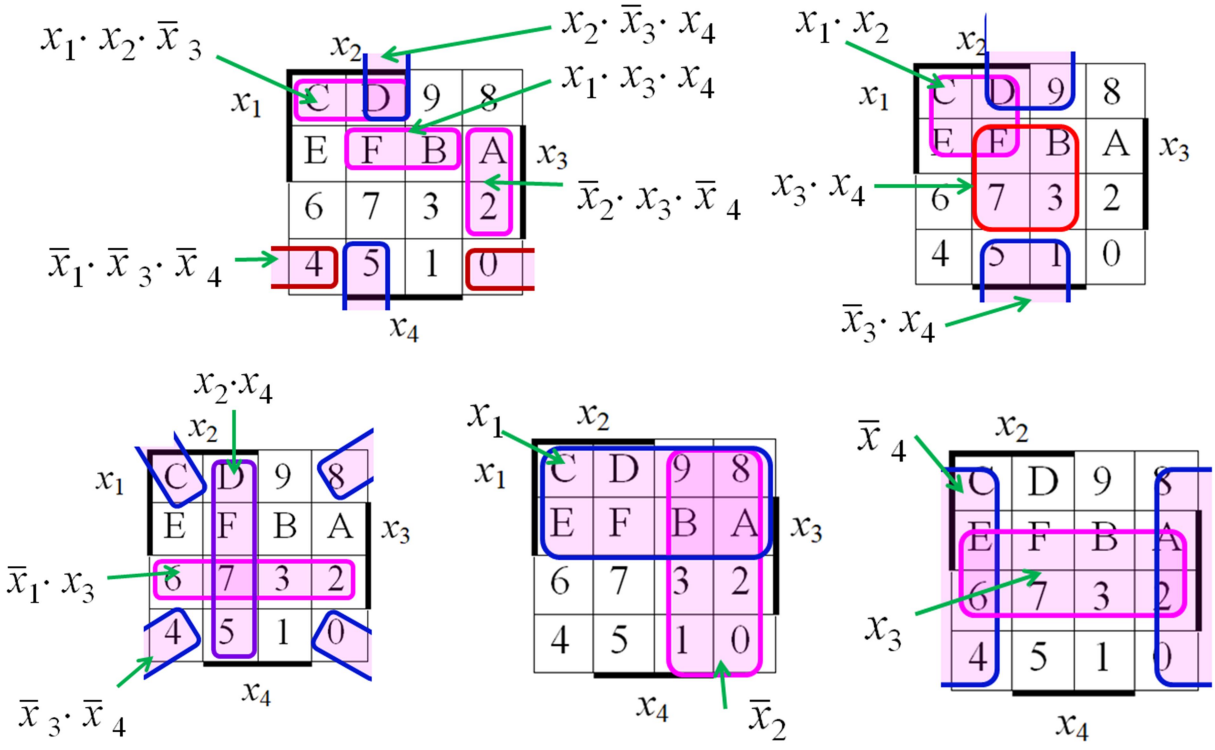


Рис. 3.9. Приклади формування мінтермів на карті Карно

Тупиковій ДНФ відповідають склеювання, в яких кожен мінтерм не містить зайвих покриттів на карті Карно. Мінімальній ДНФ відповідає склеювання з найбільш ефективним розподілом покриттів контурами.

Приклад. Дана функція: $F(a,b,c,d) = F_m(0,1,2,3,7,9,14,15)$. Занесемо цю функцію на карту Карно і зробимо всі можливі склеювання (Рис. 3.10, а). Склеюються мінтерми, охоплені контурами.

З Рис. 3.10, а видно, що склеювання призводять до утворення чотирьох імплікант, що покривають всі вісім мінтермів вхідної функції. Таким чином, диз'юнкція отриманих імплікант дає єдину тупикову ДНФ, яка має вигляд:

$$F(a, b, c, d) = \bar{a}\cdot\bar{b} \vee a\cdot b\cdot c \vee b\cdot c\cdot d \vee \bar{b}\cdot\bar{c}\cdot d \ .$$

Функція, яка представлена як ДКНФ, має таку саму карту Карно. Тобто, в карті нулі ставляться в тих позиціях, які відповідають макстермам. Однак прості імпліканти КНФ знаходяться при обводі контуром сусідніх нульових осередків карти.

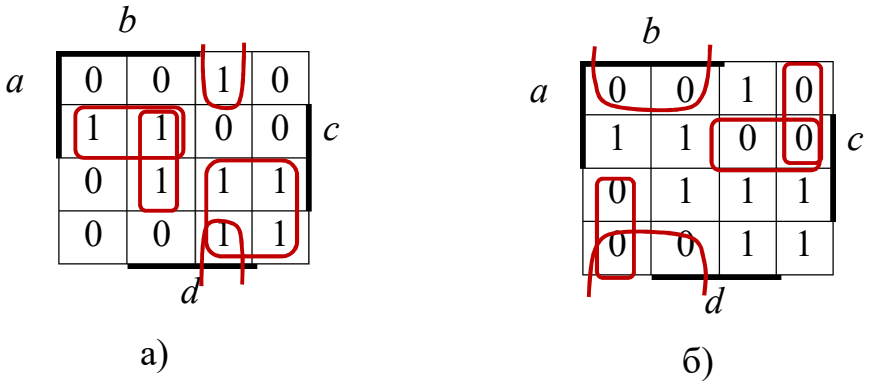


Рис. 3.10. Мінімізація ДДНФ (а) та ДКНФ (б) за допомогою карт Карно

Приклад. Та сама функція $F_m(0,1,2,3,7,9,14,15)$ дорівнює СКНФ $F_M(4,5,6,8,10,11,12,13)$, а її карта Карно з виділеними імплікантами показана на Рис. 3.10, б. Кон'юнкція отриманих імплікант дає тупикову КНФ, яка має вигляд:

$$F(a,b,c,d) = (\bar{b} \vee c)\cdot(\bar{a} \vee b \vee \bar{c})\cdot(\bar{a} \vee b \vee d)\cdot(a \vee \bar{b} \vee d).$$

Отримана мінімальна КНФ має таку саму складність, як і мінімальна ДНФ — $\Theta = 11$. Таким чином, для знаходження найпростішої формули, що виражає булеву функцію, слід розглядати мінімізацію як ДНФ, так і КНФ, серед яких потрібно вибирати оптимальну.

3.5.4. Мінімізація частково визначених булевих функцій

У реальних задачах часто трапляється так, що булева функція не визначена на деяких наборах, тобто, це ***частково визначена булева функція***. Наприклад, є такі обставини, коли не зустрічаються комбінації змінних, які відповідають невизначеним наборам. Тому ці набори називають ***надлишковими*** або ***забороненими***. Наприклад, якщо функція $F(a, b, c, d)$ обробляє набори (a, b, c, d) як коди двійково-десяткових цифр, то такі набори, як (1010), ..., (1111) не зустрічаються, так як вони заборонені. І тому на цих наборах функція може приймати довільні значення. Цю обставину можна використовувати для додаткової мінімізації частково визначеної логічної функції. У цьому випадку довизначення функції було б доцільно виконати таким чином, щоби її мінімальна нормальна форма мала б найменшу складність.

Наступний алгоритм виконує пошук мінімальної ДНФ частково визначеної функції.

1. Знайти довільним способом скорочену ДНФ функції, яка довизначена одиницями на усіх невизначених наборах.

2. Вибрати таку мінімальну ДНФ, яка покриває лише ті мінтерми, які є визначеними початково.

Аналогічно з довизначенням нульовими наборами можна скласти алгоритм для пошуку мінімальної КНФ.

Приклад, функція $F(a,b,c,d)$ невизначена на наборах (1010), ..., (1110), а на наборах (0100), (0110), (1000), (1001) приймає значення 1 Тоді вона може бути представлена картою Карно, як на Рис.3.11, а, де знак «–» означає надлишковий набір.

На Рис. 3.11, а показане покриття функції без урахування того, що вона недовизначена. В результаті отримаємо мінімізовану ДНФ:

$$F(a,b,c,d) = \bar{a}\cdot b\cdot\bar{d} \vee a\cdot\bar{b}\cdot\bar{c}.$$

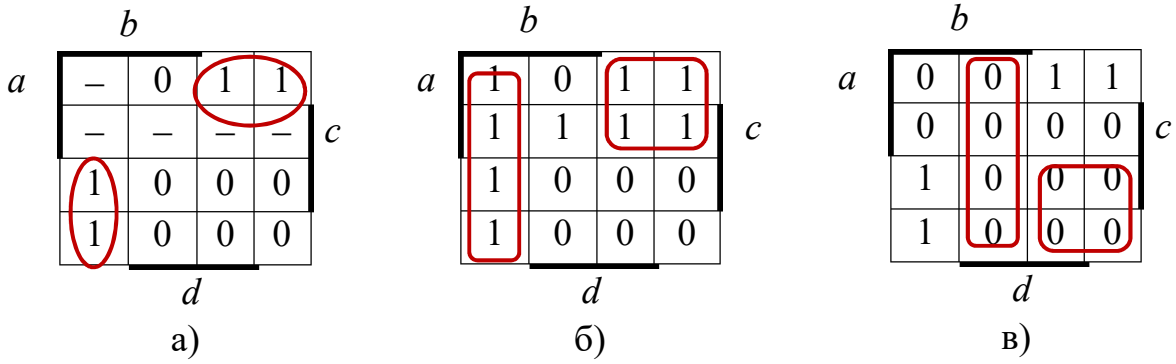


Рис. 3.11. Мінімізація логічної функції без урахування а) і з урахуванням б), в) того, що вона недовизначена

Після виконання алгоритму побудови тупикової ДНФ за недовизначеною формулою одержимо покриття як на Рис. 3.11,б і наступну ДНФ:

$$F(a,b,c,d) = b \cdot \bar{d} \vee a \cdot \bar{b},$$

яка істотно простіша за ДНФ, що одержана без довизначення.

Аналогічно може бути отримана наступна мінімальна КНФ недовизначеної функції з покриттям, яке показане на Рис. 3.11,в:

$$F(a,b,c,d) = (\bar{b} \vee \bar{d}) \cdot (a \vee b).$$

3.5.5. Абсолютно мінімальна форма представлення булевих функцій

Багато мінімальних ДНФ можна спростити. Наприклад, ДНФ $f = a \cdot b \vee a \cdot c \cdot d$ можна представити спрощено як $f = a \cdot (b \vee c \cdot d)$, але ця формула вже не є ДНФ чи іншою нормальною формою. Зате її реалізація приводить до простішої комбінаційної схеми чи програмного фрагменту. Тому є цікавою задача пошуку ***абсолютної мінімальної форми*** — форми, яка складається з мінімального числа операцій заданої функціонально повної системи.

Така задача ще не вирішена у загальному виді. Більш того, відомо, що абсолютна мінімальна форма не завжди може бути одержана з мінімальної ДНФ. У той же час, пошук форми, яка є близькою до

абсолютно мінімальної, є реальною і такою, що практично досягається.

При розробці мікросхем, як правило, користуються бібліотекою елементарних компонентів, яка притаманна даній технології. Така бібліотека складається, наприклад, з елементів І-Ні, Або-Ні, І-Або-Ні, Виключне Або, інвертор, які мають кількість входів від двох до восьми. На Рис. 3.12 показані графічні зображення у стандарті ANSI елементів типової мінімальної бібліотеки таких компонентів. Вона складається з елементів Ні, 2-, 3-, 4-входових елементів І-Ні з інверсіями на входах, 2-, 3-входових елементів Виключне Або та двовходовий мультиплексор (зображений квадратом).

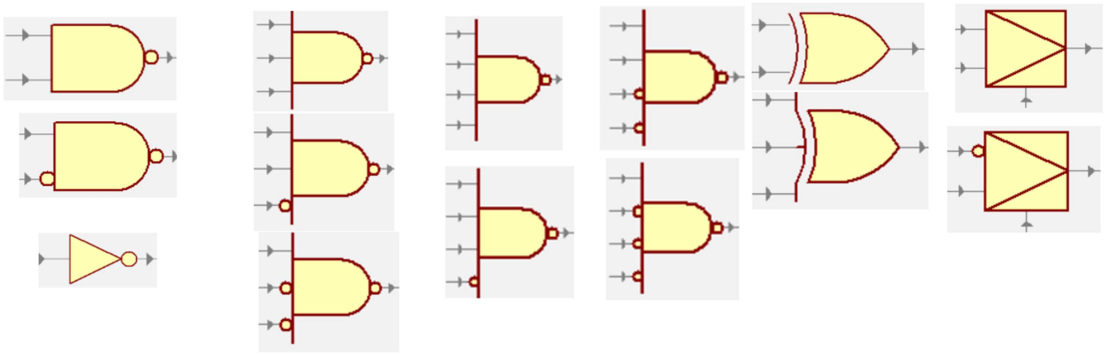


Рис. 3.12. Типова бібліотека компонентів для синтезу логічних схем

На основі такої бібліотеки в рамках САПР мікросхем створюють множину комбінаційних схем з бібліотечних елементів, які реалізують абсолютну мінімальну форму довільної функції для числа змінних 4–6. Причому одна така схема реалізує один ***клас булевих функцій***, за визначенням Шенона, які перетворюються одна в іншу перестановкою аргументів. Таким чином, при необхідності апаратної реалізації заданої функції вона не мінімізується а її оптимальна реалізація береться зі заздалегідь підготовленої таблиці.

Існує також задача ***факторизації*** булевої функції, тобто, розкладання складної функції на простіші компоненти. Часто вона називається

задачею дужкової мінімізації, яка вирішується, наприклад, винесенням загальних елементів за дужки. У загальному виді ця задача є невирішуваною.

Процес такої мінімізації булевої функції описується, так званим, факторним алгоритмом. Цей алгоритм можна показати на прикладі.

Приклад. Мінімальна ДНФ має вигляд

$$f = x_1 \cdot \bar{x}_2 \cdot x_6 \vee x_1 \cdot x_2 \cdot x_5 \cdot \bar{x}_6 \vee x_1 \cdot \bar{x}_2 \cdot x_4 \cdot \bar{x}_5 \vee x_1 \cdot \bar{x}_2 \cdot x_3 \cdot x_5 = A_1 \vee B_1 \vee C_1 \vee D_1,$$

де A_1, B_1, C_1, D_1 — відповідні терми. Представитиі функцію f через множину $F_1 = \{A_1, B_1, C_1, D_1\}$.

На першому кроці алгоритму шукаються усі попарні перетини елементів F:

$$A_1 \cap B_1 = x_1;\ A_1 \cap C_1 = x_1 \cdot \bar{x}_2;\ A_1 \cap D_1 = x_1 \cdot \bar{x}_2;$$

$$B_1 \cap C_1 = x_1;\ B_1 \cap D_1 = x_1 \cdot x_5;\ C_1 \cap D_1 = x_1 \cdot \bar{x}_2.$$

Далі вибирається та пара елементів множини F, перетин яких дає терм найбільшої довжини. Нехай це буде пара $\{C_1, D_1\}$. Тоді функція записується з винесенням елементів $x_1 \cdot \bar{x}_2$ за дужки по відношенню до пари $\{C_1, D_1\}$:

$$f = x_1 \cdot \bar{x}_2 \cdot x_6 \vee x_1 \cdot x_2 \cdot x_5 \cdot \bar{x}_6 \vee x_1 \cdot \bar{x}_2 \cdot (x_4 \cdot \bar{x}_5 \vee x_3 \cdot x_5).$$

У новій диз'юнкції складові також представляються елементами A_2, B_2, C_2 нової множини $F_2 = \{A_2, B_2, C_2\}$. На другому кроці алгоритму шукаються усі попарні перетини елементів множини F_2:

$$A_2 \cap B_2 = x_1;\ A_2 \cap C_2 = x_1 \cdot \bar{x}_2;\ B_2 \cap C_2 = x_1.$$

І знову вибирається така пара елементів множини, яка має максимальну довжину перетину. Тут це пара $\{A_2, C_2\}$. Так само, як на попередньому кроці, у A_2 і C_2 за дужки виділяється $x_1 \cdot \bar{x}_2$:

$$f = x_1 \cdot \bar{x}_2 \cdot (x_6 \vee x_4 \cdot \bar{x}_5 \vee x_3 \cdot x_5) \vee x_1 \cdot x_2 \cdot x_5 \cdot \bar{x}_6 .$$

Так само, на третьому кроці, позначають дві складові диз'юнкції як множину $F_3 = \{A_3, B_3\}$. Перетин елементів цієї множини є $A_3 \cap B_3 = x_1$. І остаточно одержується

$$f = x_1 \cdot (\bar{x}_2 \cdot (x_6 \vee x_4 \cdot \bar{x}_5 \vee x_3 \cdot x_5) \vee x_2 \cdot x_5 \cdot \bar{x}_6) .$$

Як результат, є абсолютно мінімальна форма, яка реалізується за допомогою 9 операцій І, Або, Ні на противагу 14 операціям мінімальної форми, від якої вона походить.

Як видно з Рис. 3.2, функція суми за модулем два при представленні як ДДНФ, не може бути мінімізована. Але її можна дещо спростити наступним чином:

$$x \oplus y = (x \vee y) \cdot (\bar{x} \vee \bar{y}) = (x \vee y) \cdot \overline{xy} .$$

Тут застосовуються по одній операції Або, Ні та дві операції І. Якщо розглянути чотиримісну операцію $a \oplus b \oplus c \oplus d$, то вона при реалізації як ДДНФ має 7 двомісних операцій І, 24 двомісних операції Або та 4 операції Ні. А при заміні трьох двомісних операцій суми за модулем два, одержимо формулу з 3 операцій Або, 6 операцій І і 3 операцій Ні.

3.5.6. Завдання

1. Побудувати ДДКФ і ДДНФ для функцій. Знайти мінімальні КНФ і ДНФ.

1) $f(x_1, x_2, x_3, x_4) = 0$ на наборах 0, 1, 2, 5, 6, 7, 9, 10, 11, 15;
2) $f(x_1, x_2, x_3, x_4) = 0$ на наборах 1, 3, 5, 6, 7, 10, 11, 13, 14, 15;
3) $f(x_1, x_2, x_3, x_4) = 0$ на наборах 0, 1, 2, 4, 7, 8, 10, 12, 13, 14, 15;
4) $f(x_1, x_2, x_3, x_4) = 0$ на наборах 1, 3, 5, 6, 7, 8, 10, 11, 12, 15;
5) $f(x_1, x_2, x_3, x_4) = 0$ на наборах 0, 1, 2, 4, 6, 8, 9, 10, 11, 13.

2. Знайти мінімальні КНФ і ДНФ для функцій з попереднього завдання за допомогою методу Квайна.

3. Знайти мінімальні КНФ і ДНФ для функцій з попереднього завдання за допомогою методу карт Карно.

4. Знайти всі тупикові та мінімальні ДНФ за допомогою карт Карно:

а) $xyz \vee x\bar{y}z \vee x\bar{y}\bar{z} \vee \bar{x}y\bar{z} \vee \bar{x}\bar{y}\bar{z}$;
б) $xyz \vee xy\bar{z} \vee \bar{x}y\bar{z} \vee \bar{x}\bar{y}z \vee x\bar{y}z \vee \bar{x}\bar{y}\bar{z}$;
в) $xyzt \vee xyz\bar{t} \vee xy\bar{z}t \vee xy\bar{z}\bar{t} \vee x\bar{y}z\bar{t} \vee x\bar{y}\bar{z}\bar{t} \vee \bar{x}yzt \vee \bar{x}y\bar{z}t \vee \bar{x}\bar{y}\bar{z}t$;
г) $xyzt \vee xyz\bar{t} \vee xy\bar{z}\bar{t} \vee x\bar{y}zt \vee x\bar{y}z\bar{t} \vee x\bar{y}\bar{z}\bar{t} \vee \bar{x}yzt \vee \bar{x}yz\bar{t} \vee \bar{x}\bar{y}z\bar{t}$.

3.6. Синтез комбінаційних схем

3.6.1. Процес синтезу комбінаційної схеми

Побудова комбінаційної схеми ґрунтується на тому, що вона взаємно однозначно відповідає моделі обчислень у вигляді логічних рівнянь. Причому кожен оператор рівняння відображається у відповідний логічний елемент комбінаційної схеми. Так само безпосередні залежності між операторами відображаються у лінії зв'язку між логічними елементами (Рис. 3.13).

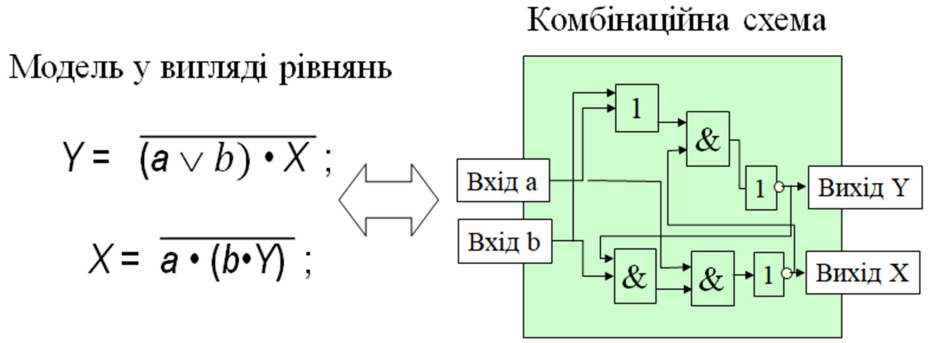


Рис. 3.13. Відображення логічних рівнянь у комбінаційну схему

Комбінаційну схему синтезують в заданому елементному базисі, наприклад, в базисі елементів І-Ні, Або-Ні або у базисі, зображеному на Рис. 3.12. Процес синтезу формально складається з двох етапів. На першому етапі формується множина ефективних схемних рішень, тобто рішень, які оптимізовані за деяким параметром. На другому етапі серед ефективних рішень вибирається рішення, яке є найкращим за заданим критерієм якості, так зване, ***оптимальне рішення***. Критерієм якості може бути складність схеми (її вартість), швидкодія або інтегральний критерій, що враховує те й інше.

Складність схеми оцінюється кількістю або ціною обладнання, що становить комбінаційну схему. У теоретичних розробках орієнтуються на довільну елементну базу і тому для оцінки витрат обладнання використовується оцінка складності схем за Квайном.

Складність за Квайном визначається сумарною кількістю входів логічних елементів у складі схеми. При такій оцінці одиниця складності — це один вхід логічного елемента. Така оцінка складності виявляється пропорційною оцінці складності схеми в числі транзисторів. Приблизно число транзисторів схеми можна оцінити в числі входів її вентилів — логічних елементів, який помножено на два (див. Рис. 3.4).

Складність схеми легко обчислюється по запису булевої функції, на основі якої будується схема: для ДНФ складність схеми дорівнює сумі кількості букв і кількості знаків диз'юнкції, яка збільшена на 1 для кожного терма виразу плюс кількість змінних, у яких є знак інверсії.

Приклад. Функція $F(a,b,d) = a\cdot\bar{b} \vee a\cdot d \vee b\cdot d$ реалізується у схемі на Рис. 3.14. Її складність за Квайном складає 6 букв, 2 знаки диз'юнкції плюс 3 і 1 інверсія — всього 12. Можна порахувати сумарну кількість входів і виходів елементів у Рис. 3.14, яка також дорівнює 12.

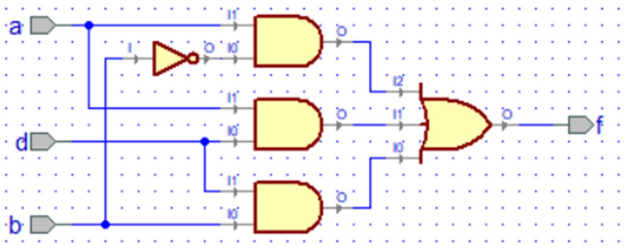


Рис. 3.14. Схема, що реалізує функцію $F(a,b,d) = a\cdot\bar{b} \vee a\cdot d \vee b\cdot d$

Швидкодія комбінаційної схеми оцінюється максимальною затримкою сигналу при проходженні його від входу схеми до виходу. Затримка сигналу кратна числу елементів, через які проходить сигнал від входу до виходу схеми. Тому швидкодія схеми характеризується значенням $r\cdot\tau$, де τ — затримка сигналу на одному елементі. Значення r визначається кількістю рівнів комбінаційної схеми. Входам комбінаційної схеми приписується нульовий рівень. Логічні елементи, пов'язані тільки з входами схеми відносяться до першого рівня. Елемент відноситься до

рівня k, якщо він пов'язаний з виходами елементів рівнів k-1, k-2, і т. д. Максимальний рівень елементів r визначає кількість рівнів комбінаційної схеми, яке називається ***рангом схеми.***

На Рис. 3.15 показана деяка комбінаційна схема, на якій виділено критичний шлях. Неважко розрахувати, що ранг схеми дорівнює 4, а довжина її критичного шляху — 4τ.

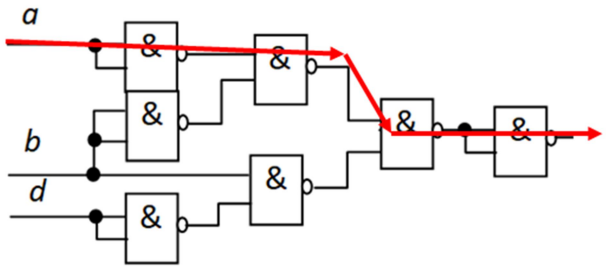


Рис. 3.15. Комбінаційна схема та виділений на ній критичний шлях

3.6.2 Мінімізація булевої функції при синтезі комбінаційних схем

Мінімізація булевої функції з метою зменшення складності схем зазвичай призводить до необхідності подання функцій у формі з дужками, якій відповідають схеми з $r > 2$. Тобто, зменшення витрат обладнання, в загальному випадку, призводить до зниження швидкодії схеми.

Для того щоб реалізувати функцію на заданій елементній базі, необхідно, крім власне мінімізації функції, прагнути також до того, щоб шукана функція була представлена за допомогою тільки операцій І-Ні (Або-Ні і т. і.).

Наприклад, маємо мінімізовану функцію трьох змінних:

$$f = a\cdot\bar{b} \vee a\cdot d \vee b\cdot d = (\bar{b} \vee d)(a \vee b)\,.$$

Логічна схема, яка реалізує цю функцію, містить 2 двовходових елементи Або і один двовходовий елемент І. Перетворимо цей вираз, використовуючи закон де Моргана:

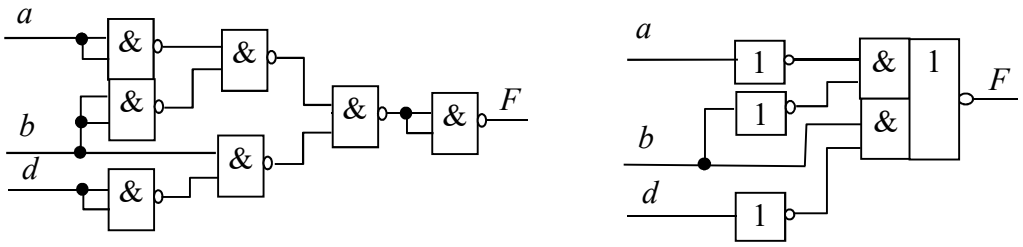
$$(\bar{b} \vee d)(a \vee b) = \overline{b\bar{d}\,\bar{a}\bar{b}} = \overline{\overline{b\bar{d}}\,\overline{\bar{a}\bar{b}}}$$

Структурна схема, яка реалізує цей вираз, представлена на Рис. 3.16, а.

Якщо базовим елементом служить елемент І-Або-Ні, то при мінімізації за допомогою карт Карно часто виявляється більш доцільним знайти КНФ. Тоді її приведення до базису І-Або-Ні виглядає так:

$$(\overline{b} \vee d)(a \vee b) = \overline{\overline{b}\overline{d}\,\overline{a}\overline{b}} = \overline{b\overline{d} \vee \overline{a}\overline{b}}\,.$$

Одержаний вираз реалізований лише одним елементом І-Або-Ні (Рис. 3.16 б).



а) б)

Рис. 3.16. Варіанти реалізації комбінаційної схеми

Як правило, схеми, які побудовані відповідно до мінімальної ДНФ функції, не є мінімальними щодо числа, як використовуваних елементів, так і їх входів. Дуже часто для подальшого спрощення схеми слід виносити за дужки загальні члени в ДНФ, тобто, застосовуючи дужкову форму, як показано у підрозділі 3.5.

3.6.3. Синтез арифметичних схем

Двійкові числа у доповнюючому коді

В усіх сучасних мікропроцесорах від'ємні числа представляються у доповнюючому коді (позитивні — у прямому коді). Доповнюючі коди мають таку назву через те, що два таких двійкові числа однакові за абсолютною величиною але з різними знаками дають при додаванні число 2^k, отже, кожне з них доповнює інше до 2^k, де k — їхня розрядність.

Наприклад,

$$A = 0101_2 = 5, \text{ та } -A = 1011_2 = -5; \quad A + (-A) = 10000_2 = 2^4.$$

Тут –A – це доповнення A до 2^4. При таких умовах найбільше чотирирозрядне число – це $0111_2 = 7$, а найменше — $1000_2 = -8$.

Якщо скласти два чотирирозрядні числа, що дорівнюють сім, одержим

$$7 + 7 = 0111_2 + 0111_2 = 1110_2 = -2,$$

хоча результат мав би бути 14. Тут ми маємо переповнення розрядної сітки.

У більшості процесорів переповнення фіксується у прапорці переповнення V = 1 (oVerflow), а також у прапорці переносу C = 1 (Carry).

Приклад. Виконуючи,

$$0011_2 + (-0111_2) = 3 - 7 = -4,$$

одержимо правильний результат

$$0011_2 + 1001_2 = 1100_2 = -4,$$

для якого V = 0, C = 0.

Також кожен процесор має прапорці нульового результату Z (Zero) та прапорець від'ємного результату N (Negative).

При множенні цілих чисел розрізняють множення без знаку (unsigned) та зі знаком (signed).

Приклад. Множення без знаку 1*15 = 15 та зі знаком 1*(–1) = –1:

$$0001_2 * 1111_2 = 00001111_2 \quad \text{та} \quad 0001_2 * 1111_2 = 11111111_2.$$

Відповідно, розрізняють команди множення та типи даних зі знаком та без знаку. Те саме стосується ділення. Розрізняють операції порівняння зі знаком та без знаку, бо, наприклад:

$$1110_2 > 0111_2 \text{ (unsigned)}, \quad \text{та} \quad 1110_2 < 0111_2 \text{ (signed)}.$$

Додавання двійкових цілих чисел у доповнюючому коді виглядає як

$$Q = B + D,$$

де $B = b_{n-1}2^{n-1} + \ldots + b_1 2 + b_0$, $D = d_{n-1}2^{n-1} + \ldots + d_1 2 + d_0$. Тоді

$$Q = (b_{n-1} + d_{n-1})2^{n-1} + \ldots + (b_1 + d_1)2 + (b_0 + d_0).$$

Суми в дужках – це порозрядні суми. Додавання починається з нульових розрядів. Результатами додавання чергових розрядів $b_i + d_i$ є розряд суми q_i та перенос у наступний розряд c_{i+1}. Таблиця 3.6 — це таблиця істинності одержання цих результатів.

Таблиця 3.6. Обчислення сигналів порозрядної суми та преносу

b_i	d_i	c_i	q_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Таблиця 3.6. задає алгоритм двійкового додавання. У реальних арифметичних пристроях розрядність внутрішніх даних часто збільшена на одиницю для фіксації випадку переповнення. Це, так званий, захисний розряд.

Приклад. Додати 8-розрядні двійкові числа –73 та 108 з використанням захисного розряду.

B = 1 1 0 1 1 0 1 1 1 = – 73
\+ D = 0 0 1 1 0 1 1 0 0 = 108
Q = 0 0 0 1 0 0 0 1 1 = 35

Тут дужкою показано дію переносу. У результаті прапорці одержали значення: V = 0, C = 1. Очевидна ознака переповнення – неоднаковість n-го та n-1-го розрядів. Тобто, V = $q_n \oplus q_{n-1}$. Але в схемах сучасних

арифметичних пристроїв для цього додатковий розряд не використовують. Замість цього, аналізують сигнали переносу і прапорець переповнення дорівнює $V = c_n \oplus c_{n-1}$.

Синтез двійкового суматора

Додавання пари розрядів b_i, d_i та переносу c_i з попереднього розряду виконується в схемі, так званого, повного суматора (full adder, FA). Послідовне з'єднання через ланцюг поширення переносу n таких схем дає схему паралельного суматора, таку, як на Рис. 3.17.

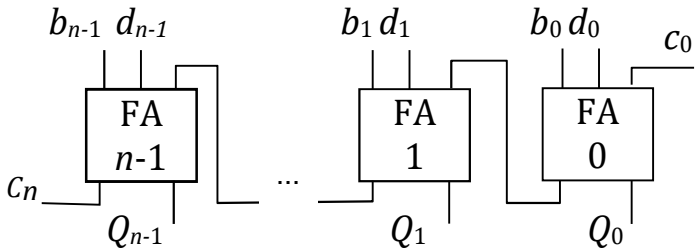


Рис. 3.17. Схема n-розрядного суматора

Внутрішня будова FA визначається схемою, яка синтезується за таблицею істинності табл. 3.6. Згідно з нею, сигнали суми і переносу обчислюються за ДНФ:

$$c_{i+1} = b_i d_i \vee b_i c_i \vee c_i d_i ;$$

$$q_i = b_i d_i c_i \vee b_i \bar{d}_i \bar{c}_i \vee \bar{b}_i d_i \bar{c}_i \vee \bar{b}_i \bar{d}_i c_i.$$

Відповідна комбінаційна схема має 10 вентилів. Оптимізовані ДНФ з урахуванням загальних імплікант

$$q_i = b_i d_i c_i \vee b_i \bar{d}_i \bar{c}_i \vee \bar{b}_i d_i \bar{c}_i \vee \bar{b}_i \bar{d}_i c_i;$$

$$\bar{c}_{i+1} = \bar{d}_i \bar{c}_i \vee \bar{b}_i d_i \bar{c}_i \vee \bar{b}_i \bar{d}_i c_i;$$

$$c_{i+1} = \bar{\bar{c}}_{i+1}.$$

Результуюча схема повного суматора показана на Рис. 3.18 і має 8 вентилів. Відповідний суматор на Рис. 3.17 називається схемою з

послідовним переносом. Він має затримку 3*n*t, де t – затримка вентиля.

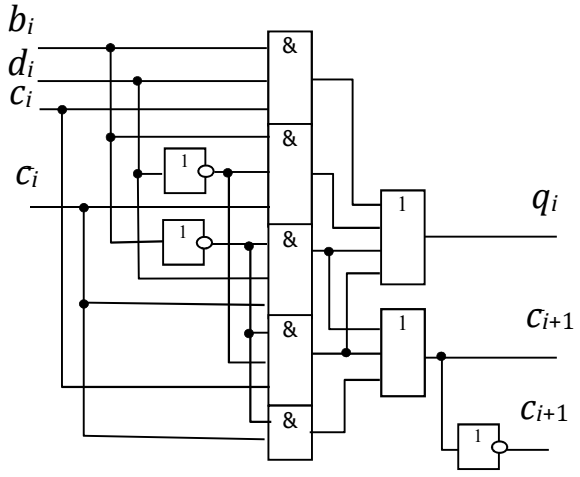


Рис. 3.18. Схема повного суматора

Можна удосконалити цю схему, якщо декомпонувати повний суматор на два напівсуматори. У напівсуматора (half adder, HA) є два входи даних, вихід суми та вихід переносу, таблиця істиності яких представлена у табл. 3.7.

Таблиця 3.7. Таблиця істинності напівсуматора

b_i	d_i	q_i	c_{i+1}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

За цією таблицею одержимо наступні мінімальні форми

$$q'_i = \overline{b_i d_i \vee \overline{b_i \vee d_i}} = b_i \oplus d_i;$$

$$c'_{i+1} = b_i d_i;$$

Тоді схема повного суматора виглядає так, як на Рис. 3.19. Схема має 7 вентилів на розряд. Затримка суматора дорівнює 2*nt,* тобто, суматор має у півтора рази більшу швидкодію.

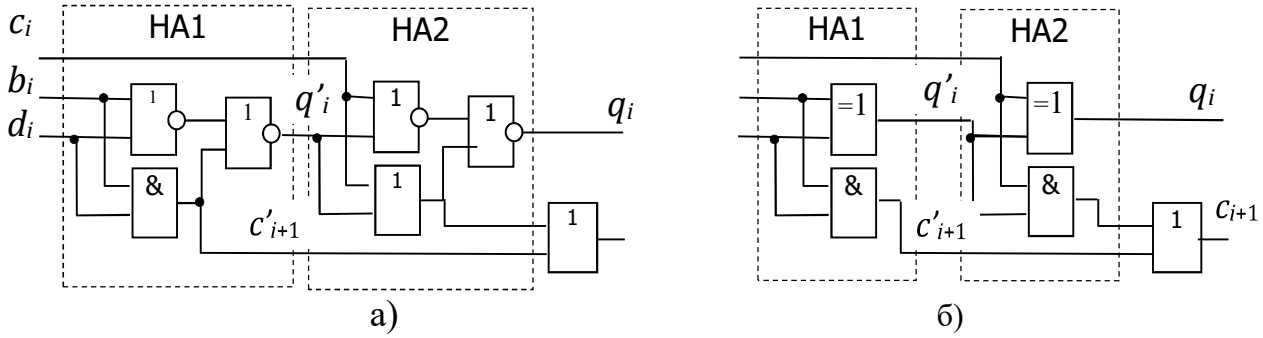


Рис. 3.19. Схеми повного суматора на елементах І, Або, Ні (а) та І, Або, Виключне Або (б)

При наявності в елементній базі елементів Виключне Або (на схемі позначений символом “=1”) варто будувати суматор саме на цих елементах (Рис. 3.19, б).

Двійковий віднімач

Операція віднімання цілих чисел в сучасних процесорах виконується за допомогою двійкового доповнення одного з операндів.

Двійкове доповнення n-розрядного числа $D \neq 0$ — це

$$\{D\}_2 = 2^n - D = (2^n - 1 - D) + 1 = (1\ldots11 - \mathrm{D}) + 1.$$

Тут вираз у круглих дужках представляє собою інверсію двійкового коду D, яка виконується інверсією усіх його розрядів. Іноді його називають ***доповненням до одного***. Тоді віднімання розраховується як

$$B - D = B + \{D\}_2.$$

Отже, щоб виконати віднімання, другий операнд інвертується та до суми додається 1 у молодший розряд.

Як правило, арифметичний пристрій виконує як додавання, так і віднімання. Причому такий суматор-віднімач будується на одному суматорі. Для цього виконуються наступні модифікації суматора:

1) Додається елемент Виключне АБО у кожен розряд доданку D_i.

2) Керуючий знаком додавання сигнал f приєднується до всіх

елементів Виключне АБО та до входу переносу молодшого розряду суматора (Рис. 3.20).

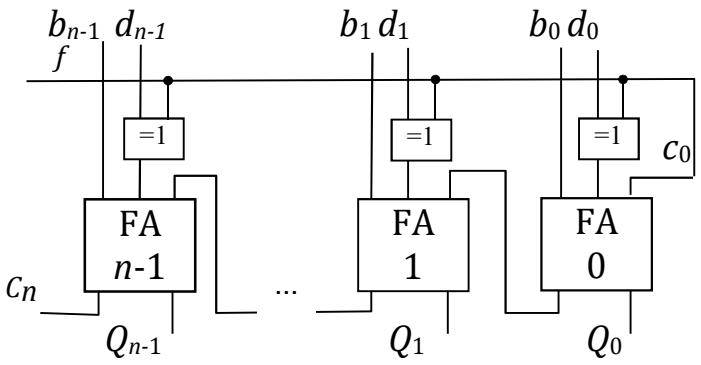


Рис. 3.20. Схема n-розрядного суматора-віднімача

При f = 0 схема працює як суматор, а при f = 1 — як віднімач. Причому при відніманні схема Виключне АБО працює як інвертор, а біт переносу С0 — як +1 до суми.

3.6.4. Дешифратор і мультиплексор

Дешифратором називається комбінаційна схема, яка реалізує усі конституенти одиниці. Ця схема має n входів та 2^n виходів, причому лише на одному з виходів встановлюється одиниця при довільному наборі аргументів.

Дешифратор адреси — це невід'ємна частини пам'яті довільного доступу будь-якої конструкції. При цьому об'єм пам'яті досягає 2^{32} і більше. Це свідчить про те, що дешифратори займають велику частку об'єму обчислювальної апаратури та їх затримка зобумовлює швидкодію блоків пам'яті.

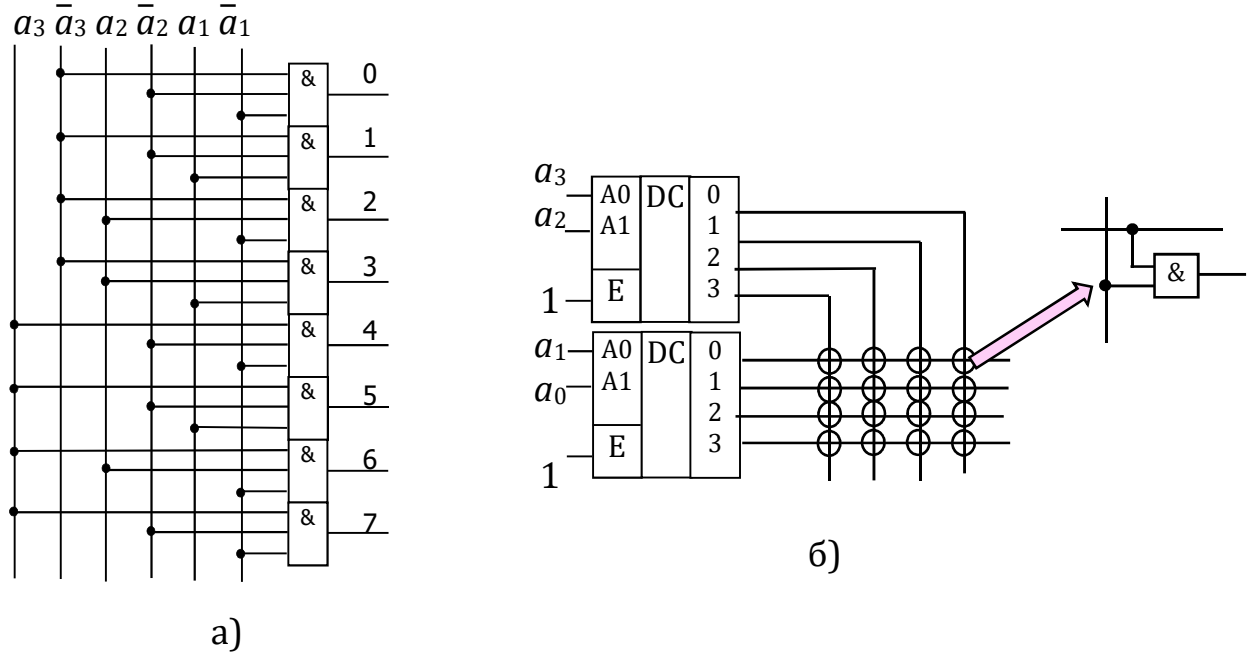
Якщо не всі з 2^n виходів задіяні, то такий дешифратор називається ***неповним***. Неповний дешифратор застосовується для декодування команд в процесорах та дешифрації адрес периферійних пристроїв та регістрів.

Найбільш поширені матричний та лінійний дешифратори. У

лінійному дешифраторі кожна конституента одиниці реалізується окремо. Цей спосіб побудови дешифраторів має перевагу у швидкодії, якщо використовуються *n*-входові схеми І, бо дешифратор виходить одноярусним. На Рис. 3.21, а показана схема лінійного дешифратора.

Матричний або прямокутний дешифратор завжди використовується для великих значень *n*. Суть цього способу побудови дешифраторів полягає в побудові двох дешифраторів для *k* та *n*–*k* змінних, виходи яких формують матрицю з 2^n вузлів. У кожному такому вузлі ставиться двохвходовий елемент І так, як показано на Рис. 3.21, б. Входи дозволу Е дешифратора дають змогу нарощувати кількість виходів дешифратора об'єднуючи кілька дешифраторів меншого розміру.

При оптимізації прямокутного дешифратора за складністю та швидкодією вибирають число *k* яке є найбільш близьким до $\frac{n}{2}$. Очевидно, що при великих значеннях *n* складність такого дешифратора можна оцінити як 2^n двовходових елементів І, бо при цьому складністю вхідних дешифраторів можна зневажати.



а) б)

Рис. 3.21. Схеми лінійного (а) і прямокутного (б) дешифраторів

Крім того, ці вхідні дешифратори також можуть бут побудовані за прямокутним способом, реалізуючи принцип ієрархії .

Схема дешифратора може бути основою для одержання довільної комбінаційної функції. Принаймні, об'єднання відповідних виходів дешифратора через елемент Або дає на виході результат ДДНФ.

Комбінаційна схема, яка має до 2^n входів даних, один вихід даних та n-розрядний керуючий вхід, який вибирає одне з вхідних даних, зазвичай називається мультиплексором. Розглянемо 2^2 = 4-вхідний мультиплексор. Тоді його булева функція дорівнює

$$y = d_0 \cdot \bar{x}_1 \cdot \bar{x}_0 \vee d_1 \cdot \bar{x}_1 \cdot x_0 \vee d_2 \cdot x_1 \cdot \bar{x}_0 \vee d_3 \cdot x_1 \cdot x_0,$$

де d_i — вхідні дані, x_1, x_0 — розряди керуючого входу. Порівнюючи це рівняння та вираз для конституенти одиниці, можна виявити, що мультиплексор являє собою комбінацію схеми дешифратора та схеми І-Або, як показано на Рис. 3.22, а. У сучасних мікросхемах схему І-Або реалізують на основі КМОН-ключів так, як показано на Рис. 3.22, б. Таким чином, виход дешифратора вибирає один з 2^n ключів, який підключає джерело сигналу до загальної точки, що грає роль "монтажної" схеми Або.

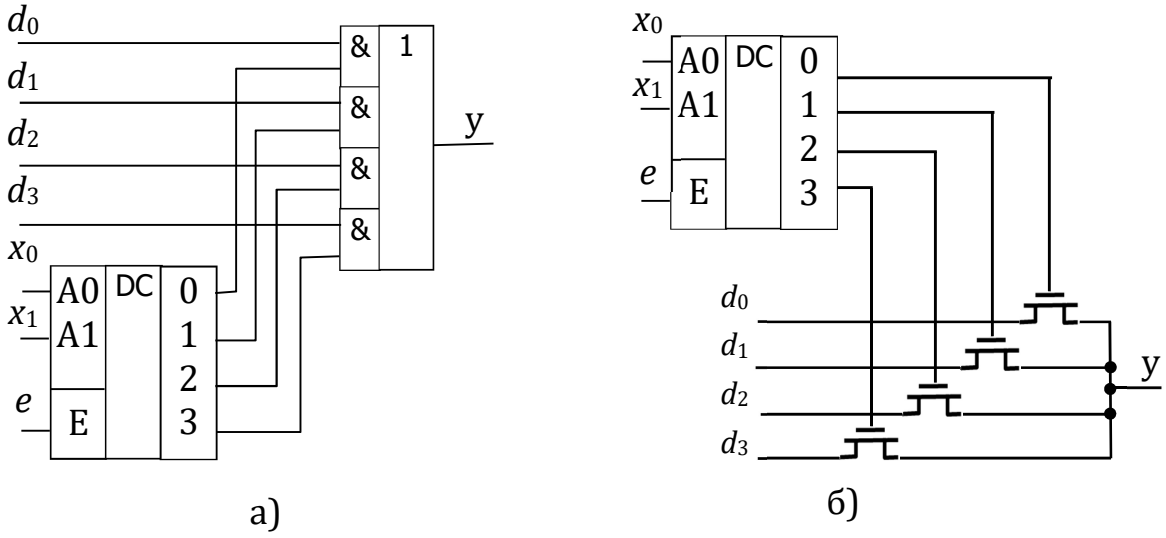


Рис. 3.22. Схема мультиплексора на логічних вентилях (а)
та на ключах (б)

У різних проєктах обчислювальних засобів мультиплексори широко використовуються для забезпечення розподілу обчислювальних ресурсів серед різних джерел даних. Наприклад, вони вставляються на входи арифметико-логічного пристрою, щоб забезпечити обробку даних, які приходять з різних напрямків.

У мікропроцесорах загальні шини для передачі даних між різними блоками виконуються на основі мультиплексорів. У цій ситуації для об'єднання у мережу n блоків процесорних елементів (PU) з можливістю пересилки даних між будь-якими блоками на кожному з входів таких блоків встановлюється n–1-вхідний мультиплексор, входи якого з'єднуються з виходами решти блоків. (див. Рис. 3.23). Через це, значна частка апаратних витрат у сучасних обчислювальних засобах припадає на мультиплексори, а швидкодія пересилки даних певною мірою визначається затримками сигналів у мультиплексорах.

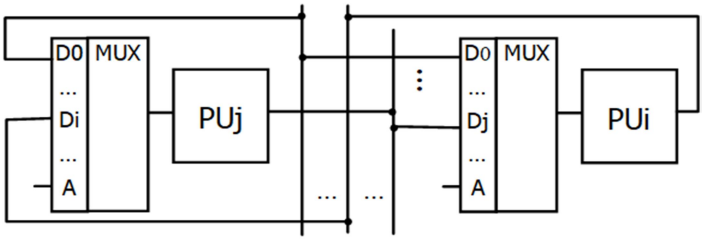


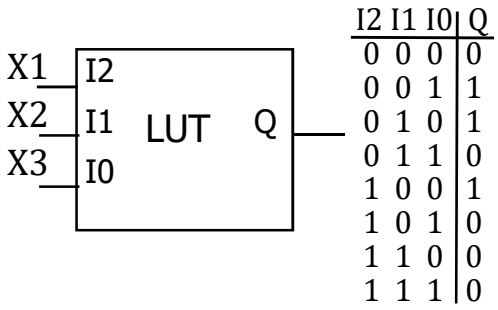
Рис. 3.23. Структура багатопроцесорної системи, система комутації яких виконана на мультиплексорах

2^n-входовий мультиплексор можна використати для одержання довільної булевої функції від n змінних. При цьому подача одиничного сигналу на i-й вхід означає кодування конституенти одиниці для i-го набору. Крім того, якщо подавати на входи мультиплексора сигнали від n+1-ї змінної чи від її інверсії, можна одержати функцію від n+1-ї змінної, так як така схема реалізує ДДНФ з мінтермами довжини n+1.

Функція дешифратора виконується у програмах за допомогою оператора **switch-case**.

3.6.4. Булеві функції на постійній пам’яті

Постійний запам’ятовуючий пристрій (ПЗП) — це запам’ятовуючий пристрій, у який запис виконується один раз протягом його використання. ПЗП може мати різну конструкцію — інформація в нього може бути занесена одноразово під час його виготовлення або кількаразово після попереднього стирання минулої інформації. В останньому випадку як ПЗП часто використовують оперативний запам’ятовуючий пристрій (ОЗП), як наприклад, у програмованих логічних інтегральних схемах (ПЛІС).



I2	I1	I0	Q
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Рис. 3.23. ПЗП логічної таблиці як елемент довільної логічної функції

Будь-який ПЗП має n адресних входів і від одного до k виходів. Двійковий набір довжини n, який подається на адресний вхід ПЗП, вибирає з нього комірку пам’яті, адреса якої визначається цим набором, а інформація з неї подається на виходи ПЗП. Схема швидкодіючого ПЗП складається з матриці носія інформації, наприклад, набору $k \cdot 2^n$ тригерів і k мультиплексорів на 2^n входів, які підключені до виходів цих тригерів. Отже, одиничний стан i-го тригера означає конституенту одиниці для i-го набору вхідних змінних, а вихід k-го мультиплексора дорівнює ДДНФ k-ї функції.

3.6.5. Завдання

1 Накреслити функціональні схеми для результатів завдання 2 з завдання 3.5.6. п.1.

2. Накреслити функціональну схему в базисі І-НІ для наступних функцій:

1) $(x \vee y)(\overline{z} \vee \overline{u}\,\overline{y})$. .

2) $\overline{x \vee \overline{y \vee \overline{z}}}$.

3) $(x\overline{y} \vee \overline{x}\,z)\overline{u \vee y\overline{z}}$.

3. Знайти мінімальну КНФ методом Вейча-Карно для наступниї недовизначених функцій та накреслити функціональну схему в базисі Або-НІ:

1) $f(x_1, x_2, x_3, x_4) = 0$ на наборах 0, 4, 7, 11, 12, 14 і недовизначена на наборах 6, 10, 13, 15;

2) $f(x_1, x_2, x_3, x_4) = 0$ на наборах 1, 5, 6, 7, 9, 14, 15 і недовизначена на наборах 2, 4, 8, 10;

3) $f(x_1, x_2, x_3, x_4) = 0$ на наборах 0, 1, 12, 14, 15 і недовизначена на наборах 2, 4, 6, 13.

4. Абстрактні цифрові автомати

4.1. Основні поняття

Цифровий або ***дискретний автомат*** — це пристрій, який виконує перетворення дискретних даних за певним алгоритмом. У загальному випадку, до дискретних автоматів належать як реальні об'єкти, такі як обчислювальні пристрої, навіть живі організми, так і абстрактні системи. Це математичні моделі у вигляді цифрових автоматів, виразів або схем, дискретних математичних структур, графів, мереж алгоритмів, тощо.

Загальну теорію цифрових автоматів поділяють на абстрактну та структурну. Відміна між ними полягає у тому, що абстрактна теорія вивчає лише поведінку автомату відносно зовнішнього середовища, не беручи до уваги їхню внутрішню будову.

У структурній теорії вивчаються способи побудови автоматів, способи кодування вхідних, вихідних сигналів та внутрішніх станів. Тобто, структурна теорія розглядає структури як автомата, так і його вхідних чинників та вихідних реакцій. Ця теорія, ґрунтуючись на абстрактній теорії автоматів та апараті булевої алгебри, дає методики розробки реальних обчислювальних засобів.

Абстрактний цифровий автомат A визначається як структура із п'яти об'єктів $A = \langle X, S, Y ; \varphi, \lambda \rangle$, де

$X = \{x_i\}$, $i = 1,\ldots,m$ — множина вхідних сигналів або вхідний алфавіт автомату A;

$S = \{s_j\}$, $j = 1,\ldots,n$ — множина станів чи алфавіт станів автомату A;

$Y = \{y_k\}$, $k = 1,\ldots,l$ — множина вихідних сигналів або алфавіт вихідних сигналів автомату A;

φ — функція переходів автомату A, яка задає відображення $(S \times X) \to S$;

λ — функція виходів автомату A, яка задає відображення $(S \times X) \to Y$

або $S \to Y$.

Множина станів S може бути скінченною або нескінченною, тобто, $|S| = \infty$. У першому випадку кажуть про ***скінченний автомат.*** Цим синонімом часто називають абстрактний цифровий автомат і він перекладається як finite state mashine (FSM).

Цифровий абстрактний автомат функціонує у ***дискретному автоматному часі***, який відмічається натуральними числами $t = 0, 1, 2, \ldots$. Часто проміжки цього дискретного часу називають ***тактами***. При цьому вважається, що переходи від стану до стану виконуються миттєво. У кожний момент часу t, тобто, у кожному такті з номером t автомат знаходиться у певному стані $s(t) = s_i \in S$.

Слід пояснити, що мається на увазі під таким об'єктом, як сигнал. ***Сигнал*** — це такий конструктивний об'єкт, який виконує три функції. По-перше, сигнал несе певне значення. Це, наприклад, біт даних, число чи інше значення з алфавіту X або Y. По-друге, сигнал використовується у процесах, які розгортаються у часі і грає роль синхронізуючого чинника — сигнал може впливати на хід виконання процесів, у даному випадку — від нього залежить наступний стан автомату $s(t+1)$. По-третє, сигнал має історію, його зміни у часі можна зафіксувати і відобразити у вигляді графіку, наприклад, $x(t)$.

Поняття ***стану*** у визначенні абстрактного автомату введене у зв'язку з тим, що автомат є моделлю для подання певного алгоритму, який розвивається у дискретному часі. Такий алгоритм на черговому кроці свого виконання у момент t повинен визначити певну дію, яка залежить від результатів, які одержані під час попереднього кроку у момент $t-1$. Отже, ці результати повинні десь зберігатись протягом одного кроку. В автоматі для цього використовується його конкретний стан $s(t)$ з множини станів S.

Наприклад, автомат перемикання світлофора за сигналом

«перемкнути світлофор» для видачі правильного вихідного сигналу — кольору світлофора — крім цього сигналу має приймати до уваги свій попередній стан, який несе інформацію про наявний колір світлофора у даний момент *t*. Ця інформація якраз забезпечується властивістю зберігати стан.

Абстрактний автомат у кожен дискретний момент часу знаходиться у певному стані $s_j \in S$. При появі вхідного сигналу $x_i \in X$ автомат переходить у інший стан $s_p \in S$. Ця дія записується як вираз $s_p = \varphi(s_j, x_i)$. Функція виходів λ показує, що автомат *A*, коли знаходиться у стані s_j при появі вхідного сигналу $x_q \in X$ видає вихідний сигнал $y_k \in Y$. Це записується як вираз $y_k = \lambda(s_j, x_q)$.

4.2. Типи абстрактних автоматів

За способом формування функції виходів виділяють три типи абстрактних автоматів: автомат Мілі, автомат Мура, С-автомат. У ***автоматі Мілі*** функція виходів λ задає відображення $(S \times X) \to Y$. У ***автоматі Мура*** функція виходів λ задає відображення $S \to Y$. У ***абстрактному С-автоматі*** вводяться дві функції виходів λ_1 та λ_2, які задають відображення $(S \times X) \to Y_1$ та $S \to Y_2$ відповідно. При цьому алфавіт виходів С-автомата є $Y = Y_1 = Y_2$ або $Y = Y_1 \cup Y_2$.

Абстрактний автомат Мілі або Мура має один вхідний та один вихідний канали (Рис. 4.1, а). Довільний абстрактний С-автомат має один вхідний і два вихідні канали (Рис. 4.1,б).

Згідно з визначенням абстрактного автомату, автомат Мілі характеризується системою рівнянь:

$$y(t) = \lambda(s(t), x(t));$$

$$s(t+1) = \varphi\,(s(t), x(t));$$

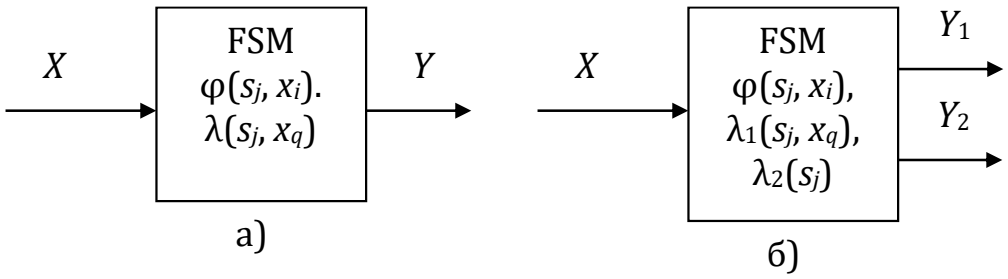


Рис. 4.1. Абстрактний автомат Мілі або Мура (а) та абстрактний С-автомат (б)

а автомат Мура — системою рівнянь:

$$y(t) = \lambda(s(t));$$
$$s(t+1) = \varphi\,(s(t), x(t)).$$

С-автомат має таку систему рівнянь:

$$y_1(t) = \lambda_1(s(t), x(t));$$
$$y_2\,(t) = \lambda_2(s(t));$$
$$s(t+1) = \varphi\,(s(t), x(t)).$$

За функціями переходів та виходів виділяють повністю визначені та часткові автомати. ***Повністю визначеним*** називається абстрактний автомат, у якого функція переходів і функція виходів визначені для усіх пар (s_j, x_i).

Частковим називається абстрактний автомат, у якого функція переходів або функція виходів чи обидві ці функції визначені не для усіх пар (s_j, x_i).

Абстрактний автомат називається ініціальним, якщо серед його станів S виділяється спеціальний початковий стан $s_0 \in S$. Отже, ***ініціальний автомат*** визначається як структура з шести об'єктів $A = \langle X, S, Y; \varphi, \lambda, s_0\rangle$. Виділення початкового стану s_0 роз'яснюється необхідністю фіксування умов початку роботи автомата при практичному його застосуванні.

Якщо автомат ініціальний, то в початковий момент часу $t = 0$ він

має знаходитись у початковому стані $s(0) = s_0$. У момент t, коли автомат знаходиться в стані $s(t)$, автомат здатний реагувати на вхідний сигнал — певну літеру вхідного алфавіту $x(t) \in X$. У відповідності до функції виходів λ автомат видає в цей самий момент часу t літеру вихідного алфавіту $y(t) \in Y$. Нарешті, у відповідності до функції переходів φ автомат переходить у наступний стан $s(t + 1)$.

Отже, якщо на вхід ініціального абстрактного автомату Мілі або Мура, який знаходиться у стані s_0, послідовно подавати літера за літерою сигнал з вхідного алфавіту $x(0)$, $x(1)$, $x(2)$,..., тобто, вхідне слово, то на виході автомату будуть послідовно з'являтись літери вихідного алфавіту $y(0), y(1), y(2)$,..., які формують вихідне слово.

У випадку С-автомату на його виходах з'являються дві послідовності: $y_1(0)$, $y_1(1)$, $y_1(2)$,..., та $y_2(0)$, $y_2(1)$, $y_2(2)$,..., які формують вихідне слово.

Отже, на рівні абстрактної теорії автоматів, функціонування цифрового автомату розуміється як перетворення вхідних слів у вихідні слова за певним алгоритмом. Згідно з цією концепцією, програма-компілятор представляється як цифровий автомат, в якому закодована граматика вхідної мови. Такий автомат перетворює послідовності вхідних символів тексту у послідовності символів скомпільованого тексту.

Абстрактні цифрові автомати за приведеним вище визначенням автомату називають ***абстрактними автоматами з пам'яттю***, оскільки зберігання стану принаймні протягом одного такту передбачає наявність пам'яті.

Деякі реальні процеси, якими керують автомати, не потребують для виконання своєї ролі знання історії розвитку процесу у часі. У таких автоматах вихідний сигнал визначається лише вхідним впливом на автомат. Такі автомати на абстрактному рівні визначаються за допомогою структури $A = \langle X, Y; \lambda \rangle$, де λ — функція виходів, яка задає

відображення $X \to Y$. Такий автомат називається ***абстрактним автоматом з тривіальною пам'яттю*** або ***комбінаційним автоматом***. Прикладом такого автомату може бути довільна комбінаційна схема, яка розглядалась у попередньому розділі.

4.3. Способи подання цифрових автоматів

Алфавіти *X, Y, S* автомату задаються так само, як звичайні множини, наприклад, як переліки їхніх елементів. Функції переходів φ та виходів λ задаються як функції, які розглядались у другому розділі — матрично, графічно чи аналітично. Тому будь-який абстрактний автомат може бути заданий трьома способами: матрично, графічно чи аналітично.

При ***матричному способі*** автомат подається двома матрицями: таблицею переходів та таблицею виходів. ***Таблиця переходів*** задає відображення $(S \times X) \to S$, тобто, функцію переходів φ. ***Таблиця виходів*** в залежності від типу автомата, що розглядається, задає чи відображення $(S \times X) \to Y$, чи відображення $S \to Y$, тобто, вона задає функцію виходів λ.

Таблиця переходів повністю визначеного автомату будується наступним чином. Стовпці таблиці позначаються буквами вхідного алфавіту, а рядки — буквами алфавіту станів. У клітинках таблиці на перетині стовпця з сигналом x_i та рядка зі станом s_l ставиться стан s_k, який є результатом переходу автомату із стану s_l під впливом вхідного сигналу x_i. Тобто, таким чином у таблиці фіксується істинність виразу $s_k = \varphi(s_l, x_i)$.

Приклад таблиці переходів певного абстрактного повністю визначеного автомату представлений у табл. 4.1. Цей автомат має вхідний алфавіт $X = \{x_1, x_2, x_3\}$ та алфавіт станів $S = \{s_1, s_2, s_3, s_4\}$.

Таблиця 4.1. Таблиця станів автомату

Стани	Вхідні сигнали		
	x_1	x_2	x_3
s_1	s_1	s_2	s_1
s_2	s_3	s_4	s_2
s_3	s_3	s_1	s_4
s_4	s_3	s_1	s_2

Якщо абстрактний автомат — частковий, то у клітинці таблиці його переходів на перетині x_i стовпця та рядка s_l ставиться прочерк, якщо перехід при таких умовах є невизначеним. У цьому випадку будь-який вхідний символ x_i за умови наявного стану s_l є забороненим. Приклад таблиці переходів деякого абстрактного часткового автомату подано у табл. 4.2.

Таблиця 4.2. Таблиця станів часткового автомату

Стани	Вхідні сигнали		
	x_1	x_2	x_3
s_1	s_1	s_2	s_1
s_2	s_3	–	–
s_3	s_3	s_1	s_4
s_4	–	s_1	–

Вигляд таблиці переходів не залежить від типу автомату (автомат Мілі, Мура чи С-автомат). Але таблиці виходів цих автоматів мають відмінності.

Таблиця виходів повністю визначеного автомату Мілі будується наступним чином. Ідентифікація стовпців та рядків такої таблиці така сама, як у таблиці переходів. У клітинці таблиці виходів з координатами x_i, s_l ставиться вихідний сигнал y_k, який автомат видає, коли знаходиться у стані s_l при наявності вхідного сигналу x_i. Таке відношення визначається виразом $y_k = \lambda(s_l, x_i)$. Приклад таблиці виходів певного повністю визначеного автомату Мілі представлений у табл. 4.3.

Таблиця 4.3. Таблиця виходів автомату Мілі

Стани	Вихідні сигнали		
	x_1	x_2	x_3
s_1	y_1	y_1	y_2
s_2	y_3	y_3	y_3
s_3	y_2	y_1	y_3
s_4	y_2	y_1	y_3

Таблиця виходів повністю визначеного автомату Мура будується простіше: кожному стану автомату призначається свій вихідний сигнал. Приклад таблиці виходів автомату Мура представлений у табл.4.4.

Таблиця 4.4. Таблиця виходів автомату Мура

Стани	Вихідні сигнали
s_1	y_1
s_2	y_1
s_3	y_2
s_4	y_3

Очевидно, що абстрактний С-автомат задається двома таблицями виходів, перша співпадає з таблицею виходів автомата Мілі, а друга — з таблицею виходів автомата Мура.

Якщо автомат Мілі є частковим, то в деяких клітинках його таблиці може стояти прочерк, що означає відсутність вихідного сигналу. При цьому прочерк обов'язково ставиться у тих клітинах таблиці виходів, що відповідають таким самим клітинам з прочерком в таблиці переходів цього автомата. Це пов'язане з тим, що якщо у частковому автоматі Мілі є пара x_i, s_l , але немає відношення $\varphi(x_i, s_l)$, тобто, такий перехід є невизначеним, то є натуральним те, що є невизначеним і вихідний сигнал при такому неіснуючому переході. Крім того, прочерк може бути не пов'язаний з невизначеним переходом. Просто, у випадку наявного переходу жодний з вихідних сигналів не видається. Правда, відсутність вихідного сигналу

часто кодується як відповідний сигнал з множини Y, як наприклад, символ ∅. Приклад таблиці виходів часткового автомату Мілі з таблицею переходів Табл. 4.2 представлений у табл. 4.5.

Таблиця 4.5. Таблиця виходів неповністю визначеного автомату Мілі

Стани	Вихідні сигнали		
	x_1	x_2	x_3
s_1	y_1	y_1	y_2
s_2	y_3	–	–
s_3	y_2	–	y_3
s_4	–	y_1	–

Прочерки у деяких клітинах таблиці виходів часткового автомату Мура не пов'язані з прочерками у клітинках його таблиці переходів. Це визначається тим, що вихідний сигнал автомату Мура залежить лише від стану, у якому знаходиться автомат. Наприклад, таблиця виходів частково заданного автомату Мура, який поданий таблицею переходів 4.2, може бути представлена так само, як табл. 4.4, якщо у кожному свому стані автомат видає якийсь вихідний сигнал. Також це може бути табл. 4.5, якщо значення вихідного сигналу автомату для деяких станів невизначено.

На практиці таблиці переходів і виходів автомату часто суміщуються в одній таблиці, яка називається ***відміченою таблицею переходів*** автомату. Відмічена таблиця переходів автомату Мілі, який представлений таблицями 4.1 та 4.3, зведена у табл. 4.6. Відмічена таблиця переходів (табл. 4.7) часткового автомату Мура відповідає табл. 4.2 та табл. 4.3.

Таблиця 4.6. Відмічена таблиця станів автомату Мілі

Стани	Вхідні сигнали		
	x_1	x_2	x_3
s_1	s_1 / y_1	s_2 / y_1	s_1 / y_2
s_2	s_3 / y_3	s_4 / y_3	s_2 / y_3
s_3	s_3 / y_2	s_1 / y_1	s_4 / y_3
s_4	s_3 / y_2	s_1 / y_1	s_2 / y_3

Таблиця 4.7. Відмічена таблиця станів автомату Мура

Стани	Вхідні сигнали			Вихідні сигнали
	x_1	x_2	x_3	
s_1	s_1	s_2	s_1	y_1
s_2	s_3	s_4	s_2	y_1
s_3	s_3	s_1	s_4	y_2
s_4	s_3	s_1	s_2	y_3

Крім табличного опису, будь-який абстрактний автомат може бути описаний матрицею з'єднань. ***Матриця з'єднань*** абстрактного автомату — це квадратна матриця розмірами $|S|\times|S|$. Кожен стовпчик і рядок матриці позначений відповідним символом стану автомату. Якщо автомат ініціальний, то перші рядок і стовпчик відмічаються станом s_0. У клітинці матриці в рядку s_j та стовпці s_p ставиться вхідний сигнал x_i, якщо під його впливом автомат переходить із стану s_j в стан s_p. Якщо ця матриця кодує автомат Мілі, то поряд із символом вхідного сигналу x_i у дужках вказується символ вихідного сигналу y_k, який автомат видає, виконуючи перехід $s_p = \varphi(s_j, x_i)$. Якщо є один перехід, але за кількома різними сигналами і з різними вихідними сигналами, то ці сигнали перелічуються у клітинці через знак диз'юнкції. Матриця з'єднань автомата Мілі, який заданий табл. 4.6, показаний у табл. 4.8.

Табл. 4.8. Матриця з’єднань автомату Мілі

	s_1	s_2	s_3	s_4
s_1	$x_1(y_1) \vee x_3(y_2)$	$x_2(y_1)$	–	–
s_2	–	$x_3(y_3)$	$x_1(y_1)$	$x_1(y_3)$
s_3	$x_2(y_1)$	–	$x_1(y_2)$	$x_3(y_3)$
s_4	$x_2(y_1)$	$x_3(y_3)$	$x_1(y_2)$	–

Якщо матрицею з’єднань кодується автомат Мура, то вихідні сигнали ставляться у тих самих рядках, які помічені відповідними станами автомату.

При ***графічному способі*** подання абстрактний автомат представляється орієнтованим графом — ***графом автомату***. Стани автомата у такому графі представляються вершинами, а переходи між ними — дугами між відповідними вершинами. Кожній такій дузі приписується вхідний сигнал x_i, яка вказує, що цей перехід виконується лише при появі цього сигнала. У вершинах графа ставиться буква s_j, яка означає стан автомата. Через те, що вершини зображаються великими кружечками, у світі граф автомата традиційно називають «пузирчастим» графом (bubble diagram). Якщо зображують графом автомат Мілі, то вихідні сигнали y_k представляються на його дугах поряд із буквою вхідного сигналу x_i. На Рис. 4.2 показаний граф автомата Мілі, який задано таблицями 4.1 та 4.3.

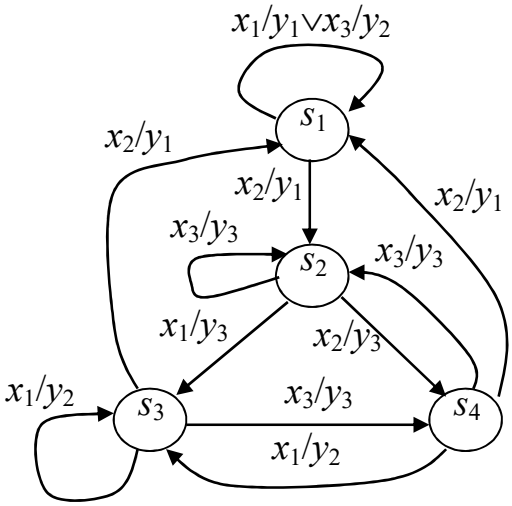


Рис. 4.2. Граф автомата Мілі

Якщо граф зображує автомат Мура, то вихідні сигнали автомату проставляються у вершинах графа у відповідності з таблицею виходів автомату. На Рис. 4.3 показано граф автомата Мура, який задано таблицями переходів і виходів 4.2 і 4.4.

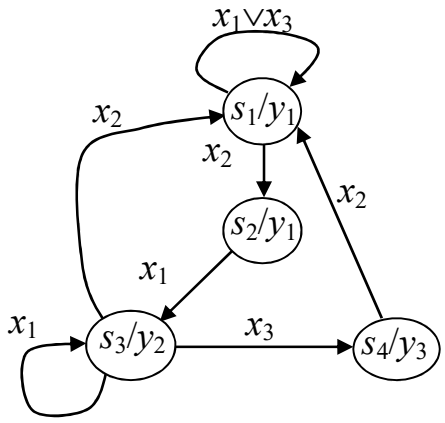


Рис. 4.3. Граф автомата Мура

У практиці комп'ютерної інженерії розглядаються лише ***детерміновані автомати.*** У таких автоматів виконується умова однозначності переходів: автомат, що знаходиться у певному стані, під впливом довільного вхідного сигналу не може перейти більш ніж в один стан.

Автомат, який задано таблицею переходів, є завжди детермінованим, оскільки на перетині стовпця x_i та рядка s_j таблиці переходів записується лише один стан $s_p = \varphi(s_j, x_i)$, якщо перехід є визначеним та риска, якщо він невизначений.

При графічному способі подання автомату умова однозначності означає, що у графі автомату з довільної вершини не може виходити дві чи більше дуг, позначених одним і тим самим вхідним сигналом.

4.4. Еквівалентні перетворення автоматів

Еквівалентними називаються автомати, які виконують одне і те саме відображення множини слів у вхідному алфавіті в множину слів у вихідному алфавіті.

Розглянемо перетворення автомату Мура у еквівалентний йому автомат Мілі. Такий перехід зручно виконувати при графічному способі подання автомату. У цьому випадку еквівалентний автомат створюється через перенос вихідного сигналу з вершин на дуги, які з них виходять. Так, якщо у вершині s_j знаходиться вихідний сигнал y_p , то цей сигнал переноситься на її вихідні дуги.

При матричному способі подання таблиця переходів автомату Мілі співпадає з таблицею переходів автомату Мура. А таблиця виходів автомату Мілі одержується з таблиці переходів автомату Мура заміною літери стану s_j , яка знаходиться на перетині стовпця з вхідним сигналом x_k і рядка s_i на вихідний сигнал y_p, який видається автоматом Мура у стані s_i.

4.5. Автомати з булевими сигналами

Абстрактні автомати мають конкретну реалізацію при їх використанні в обчислювальній техніці. При цьому вхідними та вихідними сигналами стають дані булевого типу. На Рис. 4.4 наведено простий приклад графу С-автомату, вхідними та вихідними сигналами якого є бітові сигнали.

Якщо дуга графу позначена як $x_i/y_p \vee x_j/y_q$, це означає, що якщо на входах автомату $x_i = 1$ або $x_j = 1$, байдуже, які сигнали на інших входах, то автомат переходить до стану, на який показує стрілка дуги, а виходи $y_p = 1$ та $y_q = 1$, причому інші виходи мають нульовий сигнал. Наприклад, на графі на Рис. 4.5 стан s_k залишається тим самим, коли $\bar{x}_1 \cdot \bar{x}_2 = 1$ і він змінюється на стан s_p коли $x_1 = 1$, забезпечуючи вихідний сигнал $y_1 = 1$.

Стан s_p завжди змінюється на стан s_q , тому що єдина вихідна дуга з вершини цього стану навантажена константою 1.

Для того, щоб мати детермінований автомат, у якому наступний стан завжди однозначно визначається для кожної комбінації вхідних сигналів, для міток на дугах, які виходять з кожної вершини, повинні виконуватись наступні дві умови — обмеження:

1) Якщо f_i та f_j — це будь-яка пара вхідних міток (булевих функцій) на довільних двох дугах, що виходять з вершини стану s_k, тоді $f_i \cdot f_j = 0$, при $i \neq j$.

2) Якщо n дуг виходять з вершини стану s_k і вони мають вхідні мітки $f_1, f_2, ..., f_n$, відповідно, тоді $f_1 \vee f_2 \vee ... \vee f_n = 1$.

Перша умова гарантує, що не більше ніж одна вхідна мітка дорівнюватиме 1 на вихідних дугах, а друга умова забезпечує те, що щонайменше одна вхідна мітка дорівнюватиме 1 у будь-який момент часу. Тому єдина мітка лише на одній дузі дорівнює 1, а наступний стан буде однозначно визначений для будь-якої комбінації вхідних сигналів. Наприклад, для автомату на Рис.4.4 умови виконуються для усіх вершин.

4.6. Блок-схема алгоритму автомата

Блок-схема алгоритму автомата широко використовується як альтернатива графа автомату. Блок-схема алгоритму застосовується як для опису алгоритму при його програмуванні, так і для подання скінченного автомату. У цій схемі стан автомату представлений прямокутником — вершиною стану (блоком) (Рис. 4.5).

Вершина стану містить назву наявного стану s_k, після якої через косу риску "/" вказується необов'язковий список вихідних сигналів y_j. Вершина прийняття рішення, ***умовна вершина*** представлена символом ромба, з якого виходять дві гілки — одна відповідає істинній умові і помічається як 1, а друга — хибній і помічається 0.

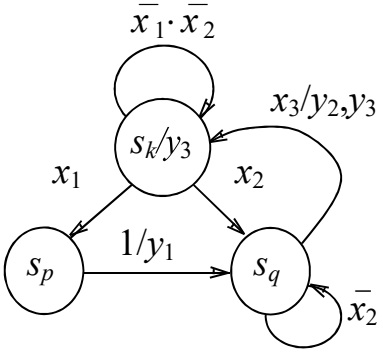


Рис. 4.4. Граф С-автомату

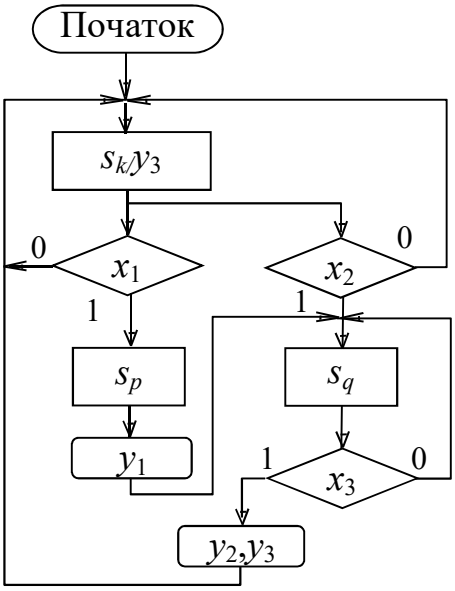


Рис. 4.5. Блок-схема алгоритму автомата

Умовою визначення того, за якою гілкою виконати перехід, є логічний вираз, який поміщається в умовну вершину, наприклад, $x_i = 1$ або просто x_i. ***Вершина умовного виходу***, яка представлена прямокутником з закругленими краями, містить список вихідних сигналів y_j, які видаються за умовою вхідних сигналів x_i у суміжній умовній вершині.

Якщо автомат — ініціальний, то додається ***вершина початку*** у вигляді овалу з написом «початок», з якої веде дуга у вершину початкового стану. Якщо автомат має зупинку, то додається ***вершина кінця*** з написом «кінець» у овалі, у яку веде дуга з вершини кінцевого стану.

Блок-схема алгоритму автомата взаємно однозначно відповідає графу автомата. При побудові блок-схеми алгоритму кожну вершину графу s_k відображають у прямокутник вершини стану зі станом s_k, до виходів якої підключається одна чи кілька умовних вершин з вхідними сигналами x_i, виходи яких відповідають дугам графу, які виходять з даної вершини. Отже, з однієї вершини стану ведуть кілька ***вихідних маршрутів*** у інші вершини стану. Якщо дана дуга графу навантажена вихідним сигналом y_j, то до відповідного виходу умовної вершини приєднують вершину

умовного виходу з сигналом y_j. Така вершина є, як правило, останньою вершиною одного з вихідних маршрутів. Отже, при виконанні алгоритму за блок-схемою, у кожен момент часу алгоритм знаходиться у одному стані s_k, тобто, активною є одна з вершин стану. При цьому видаються вихідні сигнали y_j, якщо це автомат Мура, або С-автомат.

Далі керування алгоритму проходить через умовні вершини і через єдину дугу, яка виходить з цих вершин, для якої справджуються умови. Якщо до цієї дуги під'єднано вершину умовного виходу, то автомат видає вихідні сигнали y_j так, як це виконує автомат Мілі або С-автомат. І нарешті, наприкінці такту, що розглядається, автомат переходить у наступний стан s_{k+1}, який визначається вершиною стану, куди веде маршрут з вершини стану s_k.

При побудові блок-схеми алгоритму автомату слід дотримуватися певних правил. По-перше, для кожної комбінації вхідних сигналів повинен бути визначений точно один вихідний маршрут. Це необхідно для того, щоб довільна допустима комбінація вхідних сигналів приводила лише до одного наступного стану. По-друге, зворотні зв'язки в межах одного блоку не допускаються. Так, неприпустимо з'єднувати дугою вихід та вхід однієї вершини стану без проміжної умовної вершини.

Приклад блок-схеми алгоритму автомату, яка побудована за графом автомату на Рис. 4.4, показаний на Рис.4.5.

4.7. Побудова цифрових автоматів

4.7.1 Тригер – елементарний автомат

Конкретний цифровий автомат повинен мати фізичну пам'ять. Така пам'ять традиційно виконується на тригерах. Тригер — це елементарний автомат, який має, як правило, лише два стани, що записуються як 0 та 1.

Існують тригери з більшою кількістю станів. Наприклад, комірки

деяких типів флеш-пам'яті мають 4, 8 і більше станів. Але безпосередньо в побудові автоматів вони не використовуються.

За роки розвитку комп'ютерів були розвинені методи логічного синтезу тригерних схем на основі елементарних вентилів. Але зараз ці методи не застосовуються за рядом причин. Як правило, реальні тригерні схеми збираються з транзисторів. Сучасна інтегральна технологія дійшла до такого рівня мініатюризації, що окремі транзистори перестали грати роль незалежних дискретних компонентів. Тому тригерна схема розробляється як фізичний трьохвимірний об'єкт з використанням часового моделювання електричної схеми з розподіленими параметрами.

З іншого боку, тригери, які побудовані за знаним логічним методом, виявляються непрацездатними або такими, що мають неприпустимі або непередбачувані параметри через те, що не приймаються до уваги затримки у лініях зв'язку, які в сучасних схемах переважають затримки перемикання транзисторів.

Розробники сучасних комп'ютерних схем, в тому числі автоматів, беруть готові тригери з бібліотеки компонентів конкретної інтегральної технології. На Рис. 4.6 показана схема асинхронного D-тригера, яка найчастіше використовується у мікросхемах. Її поведінка задається таблицею станів Табл. 4.9.

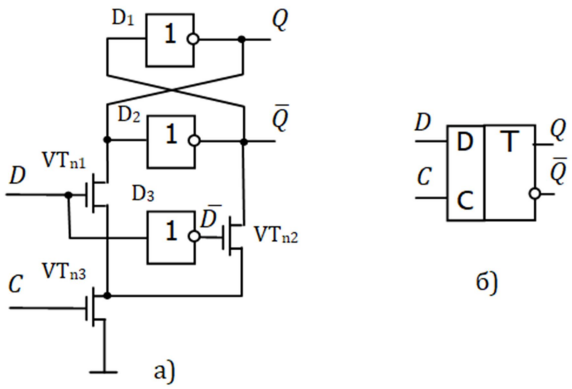


Рис. 4.6 Схема асинхронного D-тригера (а) і його умовне графічне позначення (б)

Табл. 4.9. Таблиця станів D-защіпки

D	C	Q_{i+1}
0	0	Q_i
1	0	Q_i
0	1	0
1	1	1

Асинхронний D-тригер має два основних режими роботи: режим прозорості при $C = 1$, у якому вихід тригера повторює інформацію на своєму вході D та режим зберігання при $C = 0$ у своїй внутрішній бістабільній схемі, яка має стан Q=0 чи Q=1. Через таке керування режимом зберігання асинхронний D-тригер називають ***защіпкою*** (latch). Цей тригер називають ***асинхронним тригером*** через те, що він має режим прозорості, у якому вихідний сигнал змінюється безвідносно до синхросигналу C.

Защіпки майже ніколи не використовуються як тригери в цифрових схемах зі зворотним зв'язком, таких, як цифровий автомат. У таких випадках можливе впровадження защіпок лише задіявши спеціальне завадостійке кодування станів. Як би там не було, заради гарантування стабільної роботи дискретної схеми слід застосовувати синхронні тригери, які описуються далі.

Щоб забезпечити стабільне запам’ятовування даних використовують синхронні тригери. При цьому у такому тригері немає властивості прозорості — у будь-якому режимі вихідний сигнал не повторює вхідний сигнал тригера. Зате інформація в нього записується за фронтом чи спадом синхросигналу, тобто, синхронно.

Найпростіший спосіб спроєктувати синхронний тригер — це з’єднати послідовно дві защіпки як два ступені, так як це показано на Рис.4.7, а.

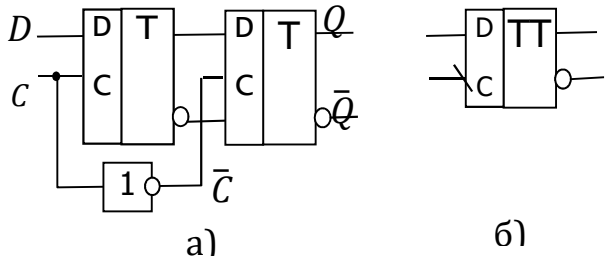


Рис. 4.7. Схема синхронного D-тригера (а) і його умовне графічне позначення (б)

Поведінка синхронного D-тригера задається таблицею станів Табл. 4.10. Перша защіпка грає роль ведучого ступеня відповідає за прийом вхідного даного. Друга защіпка є ведомим ступенем. Вона призначена для зберігання даного, яке одержане від ведучого ступеня. Для того, щоб не відбулося поширення даного зразу через два ступеня в режимі прозорості, перша і друга защіпки синхронізуються протилежними фазами синхросигналу C.

Табл. 4.10. Таблиця станів синхронного D-тригера

D	C	Q_{i+1}
0	0	Q_i
1	0	Q_i
0	1	Q_i
1	1	Q_i
0	↓	0
1	↓	1

Ведуча защіпка сприймає вхідне дане при C=1, в той час, коли ведома защіпка через те, що вона синхронізується інверсним сигналом $\bar{C}$, зберігає попереднє дане. В момент, коли синхросигнал переходить до рівня C = 0, ведуча защіпка запам'ятовує вхідне дане, а ведома защіпка стає прозорою і повторює на своєму виході дане, яке запам'ятовано. Цей тригер у англійськомовній літературі одержав назву "flip-flop" або скорочено FF, яка ассоціюється саме з такою поведінкою.

Отже, дане з ведучої защіпки переписується у ведому защіпку (і стає видимим на виході Q) у момент, коли синхросигнал переходить з $C = 1$ у $C = 0$. Через таку властивість ця схема вважається чутливою до спаду синхросигналу. Подія спаду синхросигналу показана у Табл. 4.11 символом "↓". У символі тригера на Рис. 4.7,б чутливий до спаду вхід позначається як '\'. Дві букви T у символі тригера асоціюються з тим, що всередині його є дві защіпки.

Саме синхронні D-тригери є основою для запам'ятовування станів сучасних цифрових пристроїв і зокрема, цифрових автоматів. Для зручності керування таким тригером він часто додатково має вхід R встановлення в 0 (reset), S встановлення в 1 (set) та дозволу запису E (enable).

4.7.2. Канонічний синтез цифрових автоматів

Абстрактний цифровий автомат — це зручна модель для задання великого класу алгоритмів. Тому не дивно, що ця модель почала використовуватись ще при розробці комп'ютерів перших поколінь. Перш за все, це були керуючі автомати. При цьому був розроблений та широко застосовується досі метод канонічного синтезу цифрових автоматів. Згідно з цим методом, абстрактний автомат представляється структурою, такою як на Рис. 4.8.

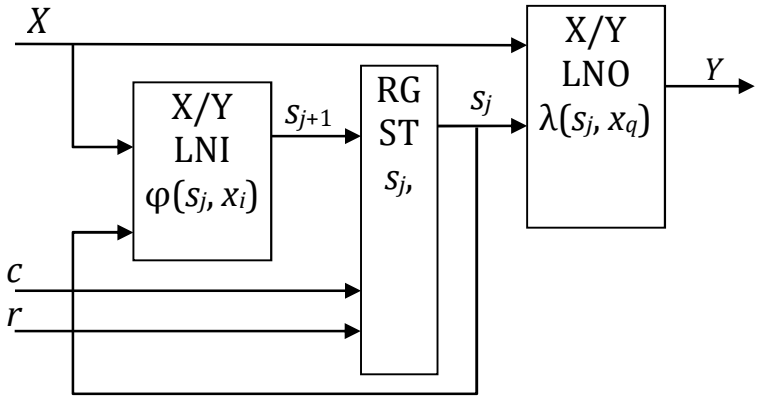


Рис. 4.8. Структура канонічного автомату

Регістр стану ST набирається з необхідного числа тригерів, які описані у попередньому підрозділі. Він зберігає наявний стан s_j принаймні протягом одного такту, який задається синхросигналом c. Комбінаційні схеми LNI, LNO, виконують обчислення функції збудження $\varphi(s_j, x_i)$ та функції виходів $\lambda(s_j, x_q)$, відповідно. Спочатку, за сигналом початкового встановлення r регістр стану встановлюється у ініціальний стан s_0. Далі в залежності від наявного стану s_j та вхідних сигналів x_i схема LNI обчислює ***сигнал збудження***, який встановлює регістр ST у наступний стан s_{j+1}. Одночасно, схема LNO виробляє вихідні сигнали y_j в залежності від наявного стану s_j та вхідних сигналів x_i.

Вхідними даними для синтезу канонічного автомату є граф абстрактного автомату чи блок-схема алгоритму, тип тригерів, з яких складається регістр станів, логічний елементний базис та критерій оптимальності, який приймає до уваги швидкодію та вартість автомату.

На першому етапі синтезу канонічного автомату вибирається кодування станів автомату. Наприклад, кількість станів $|S|$ можна закодувати двійковими кодами розрядністю $\log_2 |S|$ або унітарними кодами (одна одиниця, а решта — нулі) довжиною $|S|$. У першому випадку, одержують найпростіший регістр та мінімізовані апаратні витрати, а у другому — найбільшу швидкодію при обмеженій кількості станів $|S|$.

На другому етапі синтезуються комбінаційні схеми LNI, LNO. У найпростішому випадку вони представляють собою постійну пам’ять, яка реалізує таблиці істинності функцій $\varphi(s_j, x_i)$ та $\lambda(s_j, x_i)$. Інакше, виконується пошук мінімальних ДНФ та перетворення їх під заданий елементний базис.

Приклад. Синтезувати канонічний автомат, який виконує алгоритм на Рис. 4.5 з регістром на тригерах D-типу.

Спочатку кодуються стани як: 00, 01 і 10. У цьому випадку регістр стану — двохрозрядний з розрядами Z_2, Z_1 і може працювати як лічильник.

Далі формується таблиця станів С-автомата (Табл. 4.12.). Її колонка наступного стану має підколонки, які закодовані бітами вхідних сигналів. У цих підколонках вказується, який має бути наступний стан при наявних вхідних сигналах. Колонка вихідних сигналів має аналогічну структуру.

За даними у колонці наступного стану складають функції збудження тригерів автомата. Для D-тригерів за Табл. 4.11 виходять наступні функції :

$$D_1 = \bar{Z}_2\bar{Z}_1\,(\bar{X}_3\bar{X}_2X_1 \vee \bar{X}_3X_2X_1 \vee X_3\bar{X}_2X_1 \vee X_3X_2X_1) = \bar{Z}_2\bar{Z}_1X_1;$$

$$D_2 = \bar{Z}_2\bar{Z}_1\,(\bar{X}_3X_2\bar{X}_1 \vee \bar{X}_3X_2X_1 \vee X_3X_2\bar{X}_1 \vee X_3X_2X_1) \vee \bar{Z}_2Z_1 \vee Z_2Z_1 \vee Z_2\bar{Z}_1\bar{X}_2 =$$

$$= \bar{Z}_2\bar{Z}_1X_2 \vee Z_1 \vee Z_2\bar{Z}_1\bar{X}_2\,.$$

Слід відмітити, що стани “байдуже” при комбінаціях X_2X_1 = 11 та у стані Z_2Z_1 = 11 (для D_2) визначені як 1. Функції вихідних сигналів визначаються через дані у колонці вихідних сигналів:

$$Y_1 = Z_1; \quad Y_2 = Z_2X_3; \quad Y_3 = \bar{Z}_2\bar{Z}_1 \vee Z_2X_3;$$

Результуюча схема автомата показана на Рис. 4.9.

Табл. 4.11. Таблиця станів автомата

Наявний стан Z_2Z_1	Наступний стан, $X_3X_2X_1$=								Наявний вихідний сигнал, $X_3X_2X_1$=							
	000	001	010	011	100	101	110	111	000	001	010	011	100	101	110	111
S_k 00	00	01	10	–	00	01	10	–	Y_3							
S_p 01	10								Y_1							
S_q 10	10		00		10		00		0				Y_2, Y_3			

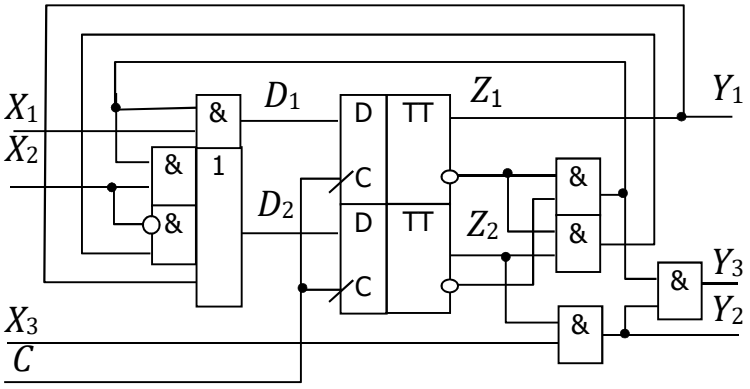


Рис. 4.9. Функціональна схема синтезованого канонічного автомату

4.7.3. Мікропрограмний автомат

До початку нашого сторіччя було поширене проєктування керуючих цифрових автоматів ***методом мікропрограмного керування***. Згідно з цим методом, для кожного стану s_j абстрактного автомату Мура вихідні сигнали y_j та сигнали, що генерують сигнал збудження s_{j+1} кодуються як, так звана, ***мікрокоманда*** (Рис. 4.10). Мікрокоманди, які формують мікропрограму, зберігаються у постійній пам'яті мікропрограм μInstruction ROM, доступ до якої виконується за лічильником мікрокоманд Sequencer. Власне, цей лічильник зберігає код s_j і грає роль регістра стану, а постійна пам'ять — служить для реалізації функції збудження та функції виходів.

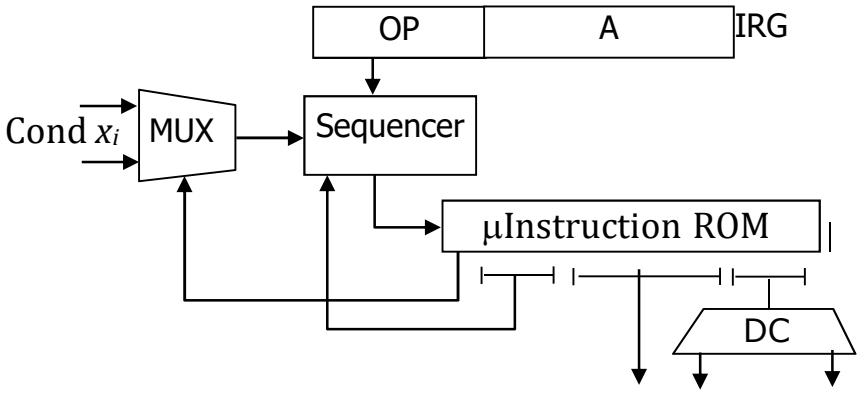


Рис. 4.10. Структура мікропрограмного автомату

Окреме поле мікрокоманди керує наступною адресою у лічильнику мікрокоманд в залежності від наявного стану s_j та вхідних сигналів Cond x_i. Таким чином, мікрокоманда з адресою — станом s_j під час свого виконання видає вихідні сигнали y_i та забезпечує вибірку наступної мікрокоманди за адресою s_{j+1} у відповідності з блок-схемою алгоритму, що виконується.

Проєктування мікропрограмного автомату полягає у розробці певного формату мікрокоманди та у формуванні мікропрограми у відповідності з заданим алгоритмом. Зручність розробки такого автомату була причиною його поширення під час проєктування комп'ютерів перших поколінь та мікропроцесорів у 80-х роках. Так, адреса потрібної мікропрограми задавалась за допомогою перекодування коду операції ОР з регістра команди IRG процесора (Рис. 4.10).

Через недоліки мікропрограмного автомату — недостатньо високу швидкодію та неможливість обробки великої кількості вхідних сигналів одночасно — мікропрограмний автомат тепер майже скрізь замінений на скінченний чи канонічний автомат. В сучасних умовах такий автомат доволі легко розробляється, використовуючи засоби САПР, як показано нижче.

4.7.4. Автоматичний синтез автомату

Розробка канонічного автомату зараз є нескладним процесом завдяки його автоматизації. Розглянемо приклад такої розробки. Для цього часто використовується мова опису цифрової апаратури, така як VHDL.

Граф автомата показано на Рис. 4.11. Нехай, цей автомат керує автоматом продажу речей. Автомат приймає монети по одній (сигнал v_1) чи дві (сигнал v_2) гривні. При прийманні чергової монети автомат видає сигнал DR (drop), який спонукає кинути наступну монету у автомат. Мета алгоритму — назбирати суму в 5 гривень (вихідний сигнал ОК).

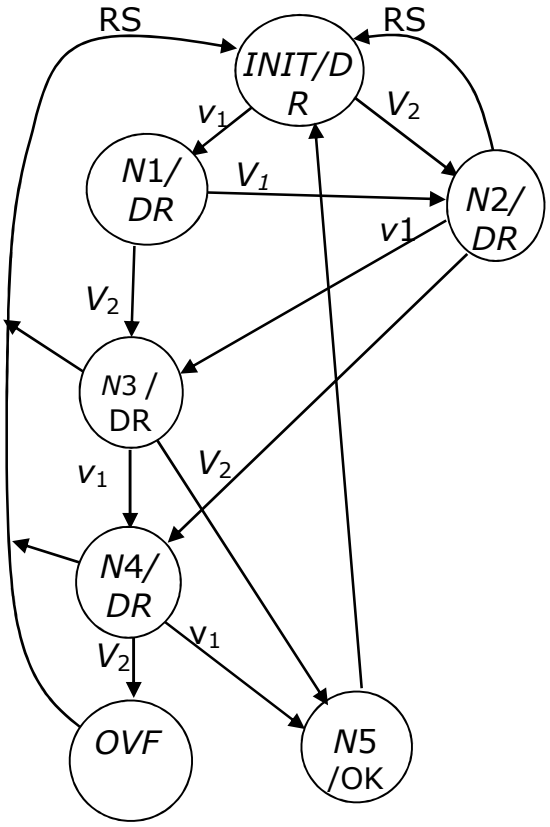


Рис. 4.11. Граф автомата, що синтезується

При збиранні неправильної суми автомат зупиняється і видає сигнал ERR. Стан OVF свідчить про те, що набрана сума є некоректною.

При описі автомату мовою VHDL спочатку описується його інтерфейс — те, як автомат виглядає зовні, тобто, це опис його вхідних та вихідних сигналів:

```
entity FSM is
 port(CLK:in BIT; -- синхросигнал
      RST:in BIT;  -- сигнал початкового встановлення
      V1:in BIT;-- кинута монета 1 грн
      V2:in BIT;-- кинута монета 2 грн
      DR:out BIT;-- "команда: киньте монету"
      OK:out BIT;-- сума досягнута
      ERR: out BIT);-- сума неправильна
end FSM;
```

Поведінка автомата описується в розділі архітектури. Тут у декларативній частині описується тип станів автомату — STATE = {INIT,

N1, N2, N3, N4, N5, OVF}, та сигнал стану st. У описовій частині в операторі процесу описується поведінка автомату. При приході сигналу RST автомат встановлюється у початковий стан INIT. При приході інших сигналів — V1 та V2 автомат змінює свій стан, причому за фронтом синхросигналу CLK. Власне, граф автомату описується у операторі CASE. У кожній з його альтернативних гілок, які визначаються наявним станом st автомату, описується те, у який стан повинен перейти автомат під впливом вхідних сигналів.

```
architecture BEH of FSM is
        Type STATE is  (INIT, N1, N2, N3, N4, N5, OVF);--states of the FSM
        signal st : STATE;
begin
        STATES:  process(CLK,RST)      -- процес опису переходів автомату
        begin
                if RST='1' then
                        st<=INIT;
                elsif Rising_Edge(CLK) then
                        case st is  --наступний стан залежить від стану st та вхідних змінних
                                when INIT => if V1='1' then st<=N1;
                                                elsif V2='1' then st<=N2;
                                                end if;
                                when N1 => if V1='1' then st<=N2;
                                                elsif V2='1' then st<=N3;
                                                end if;
                                when N2 => if V1='1' then st<=N3;
                                                elsif V2='1' then st<=N4;
                                                end if;
                                when N3 => if V1='1' then st<=N4;
                                                elsif V2='1' then st<=N5;
                                                end if;
                                when N4 => if V1='1' then st<=N5;
                                                elsif V2='1' then st<=OVF;
                                                end if;
                                when N5 => st<=INIT;
                                when others => null; --при інших станах – нічого
                        end case;
```

```
            end if;
        end process;
-- вихідні сигнали
        DR<='1' when st=INIT or  st=N1 or  st=N2 or st=N3 or st=N4 else '0';
        OK<='1' when st = N5 else '0';  -- сума досягнута
        ERR<='1' when st=OVF else '0';-- сума неправильна
end BEH;
```

Такий опис подається на вхід компілятора-синтезатора САПР мікросхем і результатом є опис скінченного автомату, який має оптимізовані апаратні витрати, швидкодію та налаштований на реалізацію у заданому елементному базисі.

4.7.5. Завдання

1. Синтезувати схему канонічного автомату керування світлофором. Вважати, що сигнали, що задають потрібні часові періоди, поступають з зовнішнього джерела.

2. Синтезувати схему автомата, який є аналогічним до автомата на Рис. 4.11, але кінцева сума має бути 6 гривень.

3. Синтезувати схему автомата, який є аналогічним до автомата на Рис. 4.11, але застосовуються монети 1, 2, 5 гривень, а кінцева сума має бути 10 гривень.

4. Синтезувати схему автомата, який виконує циклічно переходи від стану до стану: 0000, 0001, 0011, 0110, 1100, 1000 і перехід на початковий стан 0000.

5. Синтезувати схему автомата, який знаходить у довільній вхідній послідовності бітів підпослідовності коду 01110 або 10101 і повідомляє вихідними сигналами про знаходження цих кодів.

4.8. Лінійні автомати

4.8.1. Визначення

Лінійним автоматом є автомат $A = \langle X, S, Y; \varphi, \lambda \rangle$, де функція переходів $\varphi(s_j, x_i)$ та функція виходів $\lambda(s_j, x_i)$ є лінійними функціями.

Лінійність функції φ означає, що множина станів, вхідний та вихідний алфавіти належать до певної абелевої групи з операцією типу додавання $\langle M; + \rangle$ або комутативного кільця $\langle M; \cdot, + \rangle$ та те, що $\varphi(s_j, x_i) = s_j + x_i$ або $\varphi(s_j, x_i) = \alpha \cdot s_j + \beta \cdot x_i$; $\alpha, \beta \cdot \in$. M, відповідно.

4.8.2. Група цілих чисел у комп'ютерах

Група цілих чисел передбачає, що будь-який результат $s = a + b$ має належати множині цілих чисел. Крім того, кожному числу s повинен відповідати обернений елемент s^{-1} , такий що $s + s^{-1} = e$. Тут позначка "$^{-1}$" не є степенем, а позначає оберненість.

Ціле число у комп'ютерах представляється у двійковому виді у, так званому, ***прямому коді*** так само, як двійковий набір

$$s = \gamma_{n-1} \cdot 2^{n-1} + \gamma_{n-2} \cdot 2^{n-2} + \ldots + \gamma_1 \cdot 2 + \gamma_0,$$

де n — розрядність числа, γ_i — двійкові цифри (розряди) числа.

Очевидно, що множина цілих чисел, які представляються таким чином у комп'ютерах, обмежена нулем та $2^n - 1$. Якщо додати два числа a та b, які наближаються до 2^n, то одержимо число $a + b = s < 2^n$, а саме, $a + b - 2^n$ через те, що зі старшого n–1-го розряду виникне перенос у наступний розряд, якого не існує. У цьому випадку кажуть, що трапилось ***переповнення розрядної сітки*** (див. параграф 3.6.3.).

Отже, арифметичні дії у комп'ютерах виконуються не в класичній арифметиці цілих чисел, а в арифметиці за модулем 2^n. Якщо уявити, що є наступний — n-й двійковий розряд, то він матиме вагу 2^n. У цьому

випадку образом нуля є число, у якому всі цифри $\gamma_i = 0$. А це код і числа 0, і числа 2^n. Також число 2^n сприймається як одиничний елемент e групи. Тоді, обернений елемент групи дорівнює $s^{-1} = 2^n - s$ Отже, елемент s^{-1} є доповненням елементу s до 2^n.

На основі цієї властивості формують ***двійкові доповнюючі коди*** (two's complement codes), у яких числа $0 \le s < 2^{n-1}$ представляють позитивні числа, а числа $2^{n-1} \le s < 2^n$ — негативні, тобто, s^{-1} (див. парагр. 3.6.3.). Тоді, згідно з властивістю одиничного елементу, $s^{-1} + s = 0$.

4.8.3. Цілочисельні автомати

Цілочисельний автомат — це лінійний автомат над групою або кільцем цілих чисел.

Якщо мається на увазі саме група цілих чисел, то маємо автомат, який називається ***акумулятором.*** Наступний стан акумулятора визначається як

$$s_{j+1} = s_j + x_i.$$

Якщо це кільце цілих чисел, то одержуємо ***помножувач з акумулятором*** (multiplier with accumulator, MAC). Наступний стан помножувача з акумулятором визначається як

$$s_{j+1} = s_j + \alpha_k \cdot x_i.$$

4.8.4. Акумулятор

Акумулятор на функціональних схемах позначають так, як на Рис. 4.12.

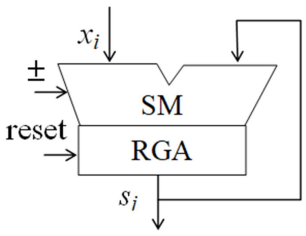


Рис. 4.12. Функціональне позначення акумулятора

Тут n-розрядний регістр RGS зберігає стан s_j акумулятора, а n-розрядний суматор SM додає до цього стану вхідне дане x_i, яке може бути як у прямому, так і доповнюю чому коді. На цьому рисунку не позначені входи синхросигналу, дозволу виконання та початкового встановлення акумулятора, які, як правило, в ньому наявні. Змінюючи у часі ці керуючі сигнали, наприклад, при їх генеруванні у відповідному автоматі Мура, акумулятор виконує певний обчислювальний алгоритм.

Як правило, лічильник команд у процесорах виконаний як акумулятор. У багатьох комп'ютерних архітектурах акумулятором називають архітектурний регістр, у який записуються результати, які одержані у арифметико-логічному пристрої.

Якщо вхідне дане є константою, наприклад, $x_i = 1$, то акумулятор називають ***лічильником*** за модулем 2^n. У загальному випадку, лічильник виконує лічбу за довільним модулем *N.* Крім того, у лічильниках використовують не тільки двійкові коди, а й багато інших, наприклад, унітарний код (одна одиниця, а інші – нулі), код Грея (сусідні двійкові комбінації відрізняються у одному розряді).

4.8.5. Помножувач з акумулятором

Помножувач з акумулятором у функціональних схемах позначають так, як на Рис. 4.13. Блок множення MPU (multiplier unit) виконує множення n-розрядних операндів з одержанням $2n$-розрядного добутку. Акумулятор на суматорі SM і регістрі RGS виконує накопичення добутків згідно з заданим алгоритмом. З метою запобігання переповнення розрядної сітки розрядність акумулятора перевищує $2n$. У реальних помножувачах з акумулятором при n =16 розрядність акумулятора досягає величин 48-64.

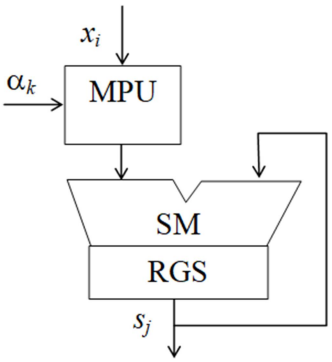


Рис. 4.13. Функціональне позначення помножувача з акумулятором

Множення з накопиченням — це типова операція в алгоритмах лінійної алгебри та цифрової обробки сигналів. Тому помножувач з акумулятором, як правило, використовується в сигнальних мікропроцесорах, а також у спеціалізованих процесорах на базі ПЛІС. Сучасні ПЛІС мають у своєму складі від десятків до кількох тисяч таких помножувачів з акумулятором. У деяких спеціалізованих процесорах для розпізнавання образів за допомогою штучного інтелекту трапляються десятки тисяч помножувачів з акумулятором.

4.8.6. Основи алгебри двійкових многочленів

Лінійним груповим кодом називається кінцева адитивна комутативна група G, елементами якої є двійкові вектори, а за операцію групи обрана операція суми за модулем 2.

Лінійно-незалежними двійковими векторами $v_1, v_2, \ldots, v_k$ називаються вектори, для яких виконується співвідношення

$$c_1v_1 \oplus c_2v_2 \oplus \ldots \oplus c_kv_k \neq 000\ldots0,$$

де $c_1 \in \{0, 1\}$ — скалярні величини, а $\oplus$ — знак операції суми за модулем 2.

Циклічні коди є різновидом лінійних групових кодів та відносяться до систематичних кодів. Першопочатково вони були створені для спрощення процедури декодування. Однак висока ефективність до виявлення помилок таких кодів забезпечила їх широке застосування на

практиці. Двійковий вектор циклічного коду для його аналізу і обробки зручно розглядати не як комбінацію нулів та одиниць, а у вигляді полінома деякого степеня

$$F(x) = a_{n-1}\, x^{n-1} \oplus a_{n-2}\, x^{n-2} \oplus \ldots \oplus a_1 x \oplus a_0,$$

де x — основа системи числення, а a_i — коефіцієнти, що належать множині {0, 1} у випадку двійкової системи числення.

Приклад. Двійковий вектор 01001 може бути поданий у вигляді полінома наступним чином:

$$F(x) = 0 \cdot x^4 \oplus 1 \cdot x^3 \oplus 0 \cdot x^2 \oplus 0 \cdot x \oplus 1 = x^3 \oplus 1.$$

Додавання многочленів зводиться до суми за модулем 2 коефіцієнтів при рівних степенях змінної x.

Множення виконується за звичайним правилом множення степеневих функцій, однак отримані коефіцієнти при даному степені додаються за модулем 2.

Ділення виконується за правилами ділення степеневих функцій, при цьому операція віднімання замінюється додаванням за модулем 2.

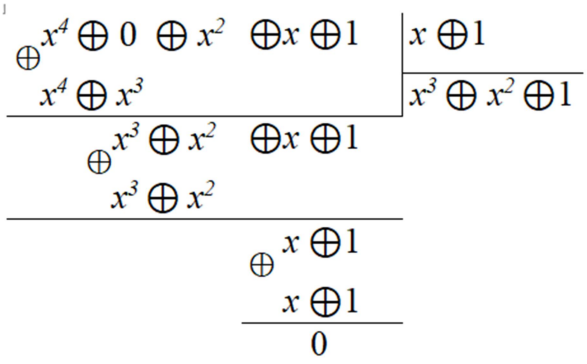
Приклад. Знайти суму многочленів $x^3 \oplus 1$ та $x \oplus 1$:

$$\begin{array}{l} 1 \bullet x^3 \oplus 0 \bullet x^2 \oplus 0 \bullet x^1 \oplus 1 \\ \oplus \\ \underline{\qquad\qquad 1 \bullet x^2 \oplus 0 \bullet x^1 \oplus 1} \\ 1 \bullet x^3 \oplus 1 \bullet x^2 \oplus 0 \bullet x^1 \oplus 0 = x^3 \oplus x^2 \end{array}$$

Приклад. Знайти добуток многочленів $x^3 \oplus 1$ та $x \oplus 1$:

$$\begin{array}{l} 1 \bullet x^3 \oplus 0 \bullet x^2 \oplus 0 \bullet x^1 \oplus 1 \\ \oplus \\ \underline{1 \bullet x^4 \oplus 0 \bullet x^3 \oplus 0 \bullet x^2 \oplus 1} \\ 1 \bullet x^4 \oplus 1 \bullet x^3 \oplus 0 \bullet x^2 \oplus 1 \bullet x^1 \oplus 1 = x^4 \oplus x^3 \oplus x^1 \oplus 1 \end{array}$$

Приклад. Виконати ділення многочленів $x^4 \oplus x^2 \oplus x \oplus 1$ та $x \oplus 1$:



Поле Галуа — це поле $GF(2^n) = \langle M; \cdot, + \rangle$, де M — множина двійкових многочленів степені n-1, "·" та "+" — операції типу множення і додавання, які обчислюються як множення і додавання з порозрядним XOR, $\oplus$, а результат є остача від ділення на незвідний многочлен степені n.

Таким чином, результати будь-якої суперпозиції операцій також належать полю $GF(2^n)$.

Поля Галуа широко використовуються у обчислювальній техніці. Це, насамперед:

— генерування псевдовипадкових кодів (далі розглядається),

— CRC-контроль в мережі Ethernet, в послідовних інтерфейсах (USB, SATA, PCIe), при зберіганні файлів на дисках, передачі цифрових сигналів і зображень через канали зв'язку (далі розглядається),

— шифрування даних, при компресії файлів (ZIP, RAR…),

— завадостійке кодування кодами Хемінга з виправленням 1, 2 помилок,

— завадостійке кодування кодами Ріда-Соломона, БЧХ з виправленням груп помилок (десятки байт).

— обчислення геш-функцій.

4.8.7. Циклічні коди

Основною властивістю циклічних кодів є наступна: якщо вектор належить множині циклічних кодів, то будь-який вектор, отриманий з

вектора, що розглядається, за допомогою циклічних зсувів, також належить цій множині.

Ідея побудови циклічних кодів базується на понятті непривідного многочлена. ***Многочлен*** називається ***непривідним***, якщо він ділиться тільки на самого себе і на одиницю, та не ділиться на жоден інший многочлен. Інакшими словами, непривідний многочлен не можна представити як добуток многочленів нижчих степенів. На непривідний многочлен без остачі ділиться многочлен $x^n \oplus 1$. Непривідні многочлени відіграють в теорії циклічних кодів роль твірних поліномів.

Приклади непривідних многочленів:

$$p(x) = x \oplus 1;$$

$$p(x^2) = x^2 \oplus x \oplus 1;$$

$$p_1(x^3) = x^3 \oplus x \oplus 1,$$

$$p_2(x^3) = x^3 \oplus x^2 \oplus 1.$$

Вектори циклічного коду будуються у відповідності до наступних правил. Нехай $Q(x)$ — будь-який двійковий вектор деякого натурального коду; x^r — одночлен степеню r; $p(x^r)$ — непривідний поліном степеню r. Тоді будь-який вектор $F(x)$ циклічного коду утворюється за допомогою співвідношення

$$F(x) = Q(x) \cdot x^r \oplus R(x^{m \le r}),$$

де $R\,(x^{m \le r})$ — остача від ділення $\dfrac{Q(x)\,x^r}{p(x^r)}$.

Таким чином, будь-який вектор циклічного коду може бути утворений множенням деякого вектора натурального двійкового коду на одночлен степеню r з додаванням до отриманого добутку остачі від ділення $\dfrac{Q(x)\,x^r}{p(x^r)}$. При побудові циклічних кодів за вказаним способом розташування інформаційних розрядів у кожному векторі коду строго

впорядковане — вони займають k старших розрядів вектору коду, а інші — молодші $r = n - k$ розрядів є перевірочними.

4.8.8. Лінійні послідовнісні автомати

Лінійний послідовнісний автомат – це лінійний автомат, який використовує функції алгебри двійкових многочленів і обробляє бітові послідовності.

Розглянемо два поширених види лінійного послідовністного автомату. Перший з них — це генератор псевдовипадкової послідовності, а другий – автомат перевірки з циклічним кодом.

Генератор псевдовипадкової послідовності побудований на основі n-розрядного регістра зсуву з лінійним зворотним зв'язком (linear-feedback shift register, LFSR). Функції наступного стану та виходу такого автомату

$$\varphi(s) = s/2 \oplus 2^{n-1} \cdot f(s),$$

$$\lambda(s) = s_0,$$

де $f(s_i)$ – функція зворотного зв'язку від розрядів коду s^i, яка визначається двійковим поліномом.

Результуюча послідовність є псевдовипадковою і формується з молодшого розряду коду s^i. ***Псевдовипадковість*** послідовності означає те, що поведінка у часі такого сигналу доволі точно нагадує поведінку випадкового сигналу, який має рівномірний розподіл ймовірності. Період послідовності, що генерується, є максимальним і дорівнює $2^n - 1$, якщо поліном зворотного зв'язку є примітивним. Умовами примітивності поліному є:

—парність відводів регістру;

—номери відводів у своїй множині є взаємно простими.

Нехай $n = 4$, тобто, $s = s_3\, s_2\, s_1\, s_0$ і примітивний поліном є $x^4 \oplus x^3 \oplus 1$.

Тоді функція зворотного зв’язку є

$$f(s) = s_2 \oplus s_0.$$

Результуючий автомат-генератор показано на Рис.4.14

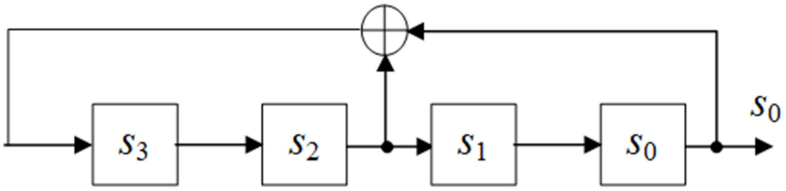


Рис. 4.14. Генератор псевдовипадкової послідовності з періодом 15

Генератори псевдовипадкової послідовності широко використовуються для діагностики та контролю обчислювальної техніки, а також у зв’язку. Наприклад, для визначення географічного положення у системі GPS задіяні псевдовипадкові послідовності з періодом $2^{10} – 1 = 1023$.

Перевірка даних з циклічним кодом (cyclic redundancy check, CRC) ґрунтується на тому, що результат згортання довгої бітової послідовності з перевірочним поліномом кардинально змінюється, якщо у цій послідовності викривити один або кілька розрядів. Таке згортання виконується за допомогою ділення вхідного двійкового поліному на породжувальний поліном, остачею якого є контрольний CRC-код.

Перш ніж переходити до опису пристрою для ділення довільного двійкового многочлена $R(x)$ на фіксований двійковий многочлен $M(x)$, розглянемо приклад.

Приклад. Нехай $R(x) = x^{14} \oplus x^{12} \oplus x^{11} \oplus x^{10} \oplus x^8 \oplus x^4 \oplus x^3 \oplus x$ і $M(x) = x^4 \oplus x \oplus 1$. Ділення дає частку, яка представляється двійковим многочленом $x^{10} \oplus x^8 \oplus x^5 \oplus x^2 \oplus x \oplus 1$ п'ятнадцятого степеня, коефіцієнтами якого є перші цифри кожного нового рядка: 0000010100100111. Остача має вигляд $0011 = x + 1$.

Таке ділення може бути реалізоване за допомогою схеми, зображеної на Рис. 4.15. Стани чотирьох тригерів пам'яті описують послідовні кроки ділення. Для даного прикладу такими послідовними станами тригерів є 0000, 0001, 0010, 0101, 1011, 0100, 1000, 0010, …, 0011. По завершенню ділення в тригерах виявляться записаними коефіцієнти остачі.

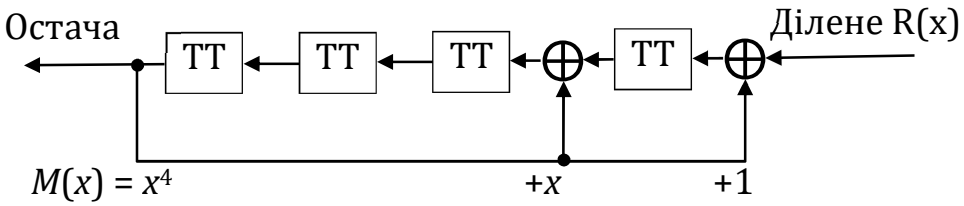


Рис. 4.15. Схема для ділення поліномів.

Обробка даних, наприклад, блоку даних, з циклічним кодом має дві окремі стадії (Рис. 4.16). На першій стадії формується контрольний код. Для цього автомат ділення поліномів встановлюється початкове значення (найчастіше – нульове) і виконується ділення. Код, який залишився у регістрі автомату, є контрольним кодом. Він дописується у кінець блоку даних. Після цього такий блок, який завершується CRC-кодом, записують у пам'ять для зберігання або посилають через комунікаційну мережу.

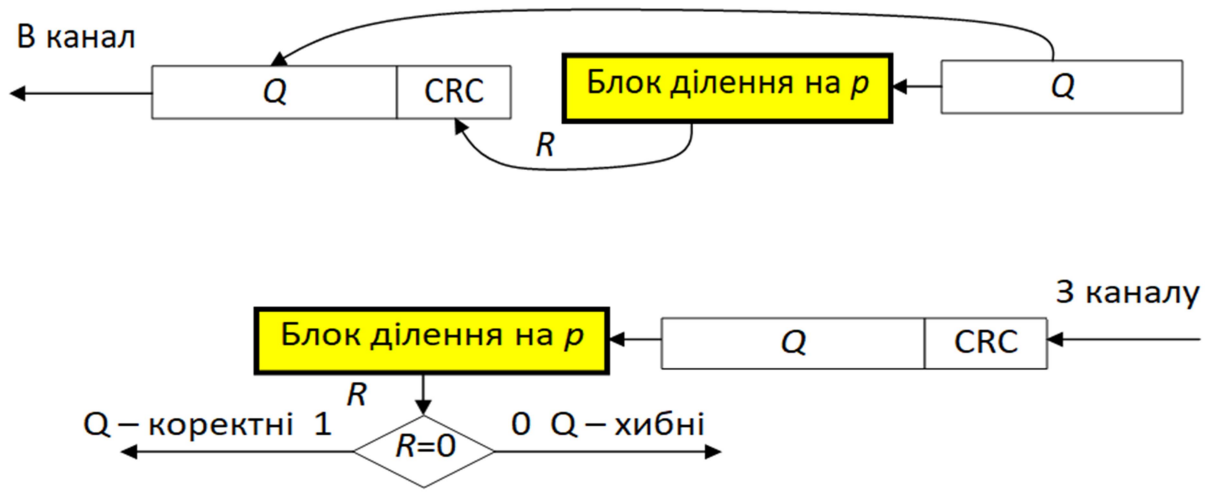


Рис. 4.16. Перетворення послідовності Q у циклічний код (а) і детектування помилки після передачі циклічного коду через канал (б)

На другій стадії блок даних з CRC-кодом, який зчитано з пам’яті чи прийнято з мережі, знову ділиться на автоматі ділення поліномів разом з самим CRC-кодом. Результатом ділення повинний бути код, який є незмінним для усіх блоків даних, що були оброблені за цим алгоритмом.

Якщо з цим блоком даних трапилась якась зміна, то добутий код буде відрізнятись від контрольного з дуже великою ймовірністю. Наприклад, для контролю файлів, які зберігаються на ЖМД, використовують 32-бітні CRC-коди, що забезпечують ймовірність неправильного декодування помилкового сектору файлу, яка менша за 10^{-9}.

CRC-кодування використовується не тільки для контролю цілісності файлів, але й пакетів даних, що пересилаються через комп’ютерні мережі. У деяких системах CRC-коди дають змогу не тільки виявляти пошкоджені дані, але й виправляти помилки.

4.8.9. Завдання

1. Маючи помножувач з акумулятором, регістри та мультиплексори, створити схему і керуючий автомат для неї, яка розраховує поліном $y = a + bx + cx^2$.

2. На основі полінома $x^5 \oplus x^2 \oplus 1$ побудувати генератор псевдовипадкової послідовності і згенерувати її.

3. Виконати ділення поліному, який задано кодом 1001 0111 0111 0011, на поліном $M(x) = x^4 \oplus x \oplus 1$.

4. Маючи поліном $M(x) = x^4 \oplus x \oplus 1$, знайти CRC-код для послідовності 0000 1111 0000 1111. Знайти остачу від ділення цієї послідовності разом з CRC-кодом на поліном $M(x)$ у випадках коли вона незмінна та коли в ній змінено перший біт на протилежний.

Рекомендована література

1. Грэхем Р., Кнут Д., Паташник О. Конкретная математика. Основание информатики. — М.: Мир. 1998. — 703 с.

2. Котов В. Е. Введение в теорию схем программ. — Новосибирск: Наука. — 1978. — 258 с.

3. Непейвода Н. Н. Прикладная логика: учебное пособие. Ижевск, изд-во Удм. ун-та, 2000. — 529 с.

4. Новиков Ф. А. Дискретная математика для программистов. Учебник для вузов. СПб.: Питер. 2009. — 384 с.

5. Самофалов К. Г., Романкевич А. М., Валуйский В. Н., Каневский Ю. С., Пиневич М. М. Прикладная теория цифровых автоматов. Київ: Вища школа, 1987. — 369 с.

6. Сигорский В.П. Математический аппарат инженера. – К.: Техніка, 1975. – 768 с.

7. Трохимчук Р. М., Нікітченко М. С. Дискретна математика у прикладах і задачах: навч. посіб. Київ. нац. ун-т ім. Тараса Шевченка. - Київ: Київський університет, 2017. – 248 с.

8. Хаггард Г., Шлипф Дж., Уайтсайдс С. Дискретная математика для программистов: учебное пособие. — М.: БИНОМ. Лаборатория знаний, 2010. — 627 с.

9. Яглом И. М. Математические структуры и математическое моделирование. М.: Советское радио, 1980. — 144 с.